Article

Attempting the Impossible: Enumerating Extremal Submodular Functions for n = 6

Elod P. Csirmaz [1] and Laszlo Csirmaz [1,2,*]

[1] Alfréd Rényi Institute of Mathematics, 1053 Budapest, Hungary
[2] Institute of Information Theory and Automation, CZ-182 00 Prague, Czech Republic
[*] Correspondence: csirmaz@renyi.hu

Abstract: Enumerating the extremal submodular functions defined on subsets of a fixed base set has only been done for base sets up to five elements. This paper reports the results of attempting to generate all such functions on a six-element base set. Using improved tools from polyhedral geometry, we have computed 360 billion of them, and provide the first reasonable estimate of their total number, which is expected to be between 1000 and 10,000 times this number. The applied Double Description and Adjacency Decomposition methods require an insertion order of the defining inequalities. We introduce two novel orders, which speed up the computations significantly, and provide additional insight into the highly symmetric structure of submodular functions. We also present an improvement to the combinatorial test used as part of the Double Description method, and use statistical analyses to estimate the degeneracy of the polyhedral cone used to describe these functions. The statistical results also highlight the limitations of the applied methods.

Keywords: vertex enumeration; double description method; submodular functions

MSC: 52B05; 52B15; 68Q25; 90C57

Academic Editors: Tiberiu Letia and Nicolae Tudoroiu

Received: 21 October 2024
Revised: 17 December 2024
Accepted: 19 December 2024
Published: 29 December 2024

Citation: Csirmaz, E.P.; Csirmaz, L. Attempting the Impossible: Enumerating Extremal Submodular Functions for n = 6. *Mathematics* **2025**, *13*, 97. https://doi.org/10.3390/math13010097

1. Introduction

Submodular functions are analogues of convex functions that enjoy numerous applications. Their structural properties have been investigated extensively, and they have applications in such diverse areas as information inequalities, operational research, combinatorial optimization and social sciences, and they have also found fundamental applications in game theory and machine learning. Consult [1] and the references therein for additional examples. For a comprehensive overview of how submodular functions are used in machine learning in particular and in optimization in general, see Bach's excellent monograph [2].

This paper reports our results in attempting to generate all extremal submodular functions defined on the subsets of a base set of size $n = 6$. Extremal functions form a unique minimal generating set of all submodular functions, thus their knowledge provides invaluable information about their structure. For $n = 5$ the complete collection of extremal submodular functions was first reported in [3]. Our aim was to develop and implement techniques which can handle the significantly more difficult problem of listing these functions for $n = 6$. Using the developed methods we succeeded in computing the first 360 billion members of the $n = 6$ list, and also in giving a reasonable estimate for their total number.

The paper is organized as follows. Sections 2 and 3 provide definitions and an introduction to submodular functions over finite base sets, and their description as a high-dimensional polyhedral cone. Section 4 provides an overview of methods from polyhedral geometry used in our computations, including the Double Description method, and the algebraic and combinatorial tests for the adjacency of rays of the cone. Section 5 introduces additional improvements to the combinatorial test.

Section 6 details our approaches to applying these tools to the $n = 6$ problem, including a description of the degree of degeneracy of the polyhedral cone, and introducing two novel orderings of the inequalities defining the cone. These orderings allowed the Double Description method to proceed further with considerably lower resource requirements. These new orders, which we named *t-opt* and *recursive*, provided additional insights into the intricate and highly symmetric structure of these polyhedral cones.

Section 6 continues with our results and statistical analyses based on two approaches to generate a high number of extremal submodular functions, while Section 7 contains an estimate for their overall number and for the total number of their (symmetrical) orbits. Finally, Section 8 summarizes our results and outlines some open questions and directions for future work.

Prior to our work, no reasonable estimate was known for the number of extremal submodular functions for a base set of size $n = 6$. Rough extrapolation from the known values for $n \leq 5$ would put their number anywhere between 10^{10} and 10^{100}. Close to the lower bound enumeration could be performed successfully with reasonable efforts, but close to the upper bound, listing the functions would be beyond every reasonable limit. Our work indicates that this number is between 10^{13} and 10^{20}, a positive result.

The main challenge addressed in this paper stems from the fact that the associated polyhedral problem is high-dimensional, and the complexity of such geometrical problems grows exponentially with the dimensions. Up to 15 dimensions, they can be routinely solved with a wide array of tools, while problems using around 30 dimensions can still be handled, but by only a handful of software packages. By contrast, enumerating the extremal submodular functions for $n = 6$ leads to a 57-dimensional geometric problem.

All charts and figures in the paper have been drawn directly from the raw data generated by the algorithms as presented. Implementations of the main algorithms discussed in this paper as well as the raw data for the figures can be found at https://github.com/csirmaz/submodular-functions-6 (accessed on 18 December 2024).

Partial results of the computations representing 360 billion extremal submodular functions in 260M orbits are available as a Zenodo Dataset at https://zenodo.org/records/13954788 (accessed on 19 October 2024). Instructions included with the GitHub repository include a quickstart guide for the scripts included, as well as for finding further rays based on the Zenodo Dataset.

2. Submodular Functions

In general, submodular functions are real-valued functions defined on some lattice. In the most important case—which is also the subject of this paper—the lattice is formed by all subsets of a (finite) set X, called the *base*, with the intersection and union as lattice operations. While definitions are spelled out for this special setting, those in general are similar. Some other lattice-based functions are discussed as illustrations.

The functions we are interested in assign real numbers to subsets of a (typically finite) base set X. Such a function f is called *modular* if

$$f(A) + f(B) = f(A \cap B) + f(A \cup B)$$

holds for all subsets A and B of X. Functions satisfying

$$f(A) + f(B) \geq f(A \cap B) + f(A \cup B) \tag{1}$$

for all A and B are called *submodular*, referring to the fact that the right hand side is below what modularity would require. Similarly, functions satisfying

$$f(A) + f(B) \leq f(A \cap B) + f(A \cup B)$$

are called *supermodular*. A function is modular if and only if it is both submodular and supermodular.

A general example of a modular function is a (possibly signed) discrete measure on X. In the more general setting X is infinite and the function is defined only on a sublattice of all subsets of X, e.g., the Lebesgue measure on the measurable subsets of the unit interval. Taking the outer measure instead, which is defined on all subsets of the base set, we also get a suitable f, but it is only submodular, see, e.g., [4]. In another important example the underlying lattice is the lattice of the vectors in the n-dimensional Euclidean space endowed with the coordinatewise minimum and maximum as lattice operations. Here the function $f : \mathbb{R}^n \to \mathbb{R}$ is submodular if

$$f(\mathbf{x}) + f(\mathbf{y}) \geq f(\mathbf{x} \wedge \mathbf{y}) + f(\mathbf{x} \vee \mathbf{y}). \tag{2}$$

Writing $\mathbf{z} = \mathbf{x} \wedge \mathbf{y}$, $\mathbf{a} = \mathbf{x} - \mathbf{z}$, $\mathbf{b} = \mathbf{y} - \mathbf{z}$, both $\mathbf{a}$ and $\mathbf{b}$ have non-negative coordinates, and (2) rewrites to

$$f(\mathbf{z}+\mathbf{a}) - f(\mathbf{z}) \geq f(\mathbf{z}+\mathbf{b} + \mathbf{a}) - f(\mathbf{z}+\mathbf{b}).$$

This formula is interpreted as the "diminishing returns property" [5]: investing (adding) $\mathbf{a}$ at state $\mathbf{z}$ yields the return $f(\mathbf{z}+\mathbf{a}) - f(\mathbf{z})$. Investing the same amount $\mathbf{a}$ after another investment $\mathbf{b}$ has been done yields a smaller return. In this context supermodularity corresponds to "accelerating returns" [6], where the same amount of investment applied later yields larger returns.

The present paper deals exclusively with the case when X is finite and functions are defined on all subsets of X. The usual notation is used: subsets of the base set X are denoted by upper case letters such as A, B, K, L, etc.; elements of X by lower case letters, e.g., i, j, k. The $\cup$ sign denoting the union of two subsets is frequently omitted as well as the curly brackets around singletons. Thus, for example, Ai denotes the subset $A \cup \{i\}$. The collection of all subsets of X, including the empty set, is denoted by 2^X.

The following simple claim summarizes the basic properties of $\mathsf{MOD}(X)$, the class of modular functions on a finite set X, see [7].

Claim 1. (a) *Choose $z \in \mathbb{R}$ and $a_i \in \mathbb{R}$ for $i \in X$ arbitrarily. Then $m : 2^X \to \mathbb{R}$ defined by $m(A) = z + \sum\{a_i : i \in A\}$ for $A \subseteq X$ is modular.*
(b) *Every $m \in \mathsf{MOD}(X)$ is of this form. Consequently,*
(c) *$\mathsf{MOD}(X)$ is an $|X| + 1$-dimensional linear space.*

Submodular functions remain submodular after adding (or subtracting) a modular function. Also, a conic (that is, non-negative linear) combination of submodular functions is again submodular. A collection $\mathcal{G}$ of submodular functions over X is a *generator* if every submodular f is the sum of a modular function and a conic combination of elements of $\mathcal{G}$. A submodular function is *extremal* if it is an element of a minimal generator set. Extremal submodular functions are determined uniquely up to a (positive) scaling factor and a modular shift, see Theorem 1. Since for a fixed finite X there are only finitely many such extremal submodular functions by Theorem 2, it is possible, at least in theory, to list all extremal submodular functions. This list also provides all extremal supermodular functions as well since f is submodular if and only if $-f$ is supermodular. Such a list would carry invaluable information about the structure of submodular functions. Several optimization techniques rely on finding an appropriate extremal submodular function [1]; with such a list, such problems would reduce to a simple search. Knowledge of all extremal functions for the case $|X| = 4$ was an essential ingredient in investigating properties of conditional independence structures in [8,9]. Extremal supermodular functions for $|X| = 5$ were reported in [3]; these functions were used in [10] to investigate and create new non-Shannon type entropy inequalities for five random variables, and in [11] to characterize conditional independence structures.

Bayesian networks [12] play an important role in machine learning. They have become a popular representation for encoding uncertain expert knowledge in expert systems. Bayesian networks form a class of probabilistic graphical models defined over a set of random variables, each describing some quantity of interest, that are associated with the nodes of a directed acyclic graph (DAG). Arcs in the graph express direct dependence relationships between the variables, while graphical separation in the graph implies conditional independence in probability. Each Bayesian network has an equivalent description using a linear condition (derived from the DAG) on the *entropies* of the subsets of the variables [13], which is a (p-standardized) *submodular* function. All such submodular functions can be obtained as a non-negative (conic) combination of those extremal submodular functions which satisfy this derived linear condition. Knowledge of these extremal submodular functions allows to quickly check whether any conditional independence statement (two sets of nodes are independent given a third one) holds necessarily in the network: it holds if and only if it holds for these extremal functions. For more on the conditional independence structures of Bayesian networks, see [14].

In a more general setting, enumerating extremal rays is an essential tool in objective space vector optimization problems [15], where several linear objectives are to be optimized simultaneously given a collection of linear constraints. Typically, there is no single optimal value; rather, the task of the optimization is to describe its *Pareto front*. It consists of those goal vectors where none of the objectives can be improved on without destroying the optimality of some other objective. In the linear, non-degenerate case the Pareto front is an unbounded polyhedron. The solution of the vector optimization problem is an enumeration of the vertices and facets of this polyhedron. Many of our techniques can also be applied in this area, especially when both the objectives and the constraints exhibit many symmetries.

In our work attempting to generate all extremal submodular functions for $|X| = 6$, the main tools are methods from high-dimensional polyhedral computations. For basic concepts and notions of this area, consult [16]; a comprehensive overview from the computational point of view is M. Fukuda's excellent monograph [17]. A comparative study of different polyhedral methods and implementations of different algorithms can be found in [18]. For notions and methods from linear algebra consult [19].

3. The Cone of Submodular Functions

Let $n = |X|$ be the number of elements in the (finite) base set X. Any real function defined on the subsets of X can be represented by a 2^n-dimensional vector r indexed by the subsets of X as

$$r = \langle r_A : A \subseteq X \rangle,$$

where r_A is the value of the function at A. Both the function and the vector notation will be used interchangeably.

3.1. Submodular Inequalities

For disjoint subsets A, B, K of X with A and B not empty, let $\delta(A, B|K)$ be the 2^n-dimensional vector with four non-zero coordinates: $+1$ at indices AK and BK, and -1 at indices K and ABK. (The assumptions on A, B, K ensure that these indices are different). The scalar product of r and $\delta(A,B|K)$ is

$$r \cdot \delta(A,B|K) = r_{AK} + r_{BK} - r_K - r_{ABK}.$$

A function represented by r satisfies all inequalities in (1), that is, r is submodular, if and only if the scalar product $r \cdot \delta(A,B|K)$ is non-negative for every choice of A, B and K. We will also use the bare triplet $(A,B|K)$ to mean the *inequality* expressing

$$f(AK) + f(BK) - f(K) - f(ABK) \geq 0$$

for some unspecified function f, while $\delta(A,B|K)$ is the vector corresponding to, or labeled by, this inequality.

The complete set of the inequalities $(A,B|K)$ is highly redundant. It can be illustrated based on the equality

$$\delta(A,B|K) + \delta(A,C|KB) = \delta(A,BC|K) \tag{3}$$

known as the *chain rule* in Information Theory [20]. From this, if the inner products with the vectors on the left hand side are non-negative, then so is with the one on the right hand side. Therefore the inequality $(A,BC|K)$ is a consequence of $(A,B|K)$ and $(A,C|KB)$. The unique minimal set of inequalities which implies all others consists of the so-called elementary inequalities (the terminology is from [14], see also [7]). The inequality $(A,B|K)$ is *elementary* if both A and B are singletons. The matrix formed from the row vectors corresponding to the elementary inequalities is denoted by $M^\sharp$:

$$M^\sharp = \{\delta(i,j|K) : i,j \in X, K \subseteq X \setminus ij\}.$$

Clearly, $M^\sharp$ has 2^n columns and $\binom{n}{2}2^{n-2}$ rows.

Claim 2. (a) *The function represented by the vector r is submodular if and only if $M^\sharp \cdot r \geq 0$.*
(b) *All elementary inequalities in $M^\sharp$ are necessary.*
(c) *$M^\sharp$ has a rank of $2^n - n - 1$.*

Proof. (a) To show that $M^\sharp \cdot r \geq 0$ implies $r \cdot \delta(A,B|K) \geq 0$ use induction on $|A| + |B|$. If $|A| + |B| = 2$, then $(A,B|K)$ is elementary, thus it is a row in $M^\sharp$. Otherwise, either A or B has at least two elements. Use the chain rule (3) and induction.

(b) To show that the row $\delta(i,j|K)$ in $M^\sharp$ cannot be omitted, it suffices to exhibit a non-submodular function g which satisfies all elementary inequalities except this one. Let $|K| = k$ (clearly $0 \leq k \leq n{-}2$), and set $g(iK) = g(jK) = k$. For all other subsets $A \subseteq X$ define $g(A) = \min\{|A|, k{+}1\}$. Then

$$g \cdot \delta(i,j|K) = k + k - k - (k{+}1) = -1,$$

and it is easy to check that for the other elementary triplets the inner product $g \cdot \delta(i',j'|K')$ is either zero or plus one.

(c) The kernel (zero set) of $M^\sharp$ is the set of modular functions as $M^\sharp \cdot m = 0$ if and only if m is modular. By point c) of Claim 1 modular functions form an $n{+}1$-dimensional linear space. $M^\sharp$ has 2^n columns, thus the rank of $M^\sharp$ is $2^n - (n+1)$, as was claimed. □

3.2. Standardization

Two submodular functions are "equivalent" if their difference is modular. This relation clearly splits the submodular functions into equivalence classes. *Standardization* is a method that assigns the same representative to each element of such a class. For supermodular functions three natural, theoretically motivated standardization methods are mentioned in Chapter 5.1 of [14]. For submodular functions, however, with a focus on matroid theory, the following *polymatroidal* standardization is recommended. Consider the following linear space of functions on 2^X:

$$\mathcal{S}_p(X) = \{f : f(\emptyset) = 0 \text{ and } \forall i \in X\; f(X \setminus i) = f(X)\}.$$

The *p-standardized* form of f is the only element which is both in the equivalence class $f + \mathsf{MOD}(X)$ and the linear space $\mathcal{S}_p(X)$.

As defined in [10] or in [21], *tight polymatroids* on X are those submodular functions which are additionally

pointed: $f(\emptyset) = 0$;
monotone: $A \subseteq B$ implies $f(A) \le f(B)$; and
tight at the top: $f(X) = f(X \setminus i)$ for all $i \in X$.

Clearly a p-standardized function g is both pointed and tight. Since monotonicity is a consequence of submodularity and these two properties, g is also monotone, thus it is a tight polymatroid. The difference of two different tight polymatroids is never modular, which proves

Claim 3. *The class of p-standardized submodular functions is the class of tight polymatroids.*

A consequence of Claim 3 is that p-standardized submodular functions are automatically non-negative. One of our algorithms sketched as Code 4 uses this fact as a quick preliminary check.

Other standardizations can be defined analogously by choosing different $(n+1)$-dimensional linear subspaces which intersect each $f+\mathsf{MOD}(X)$ class in a single element. Two of the subspaces suggested in [22] are the lower space

$$\mathcal{S}_\ell(X) = \{f : f(A) = 0 \text{ when } |A| \le 1\},$$

and the upper space

$$\mathcal{S}_u(X) = \{f : f(A) = 0 \text{ when } |A| \ge n-1\},$$

and there are many other possibilities. Using a different standardization has no, or little, effect on the computational complexity of our algorithms (however, it might affect the magnitude of the numbers to work with), thus the choice of $\mathcal{S}_p$ is rather arbitrary, and was influenced mainly by the authors' familiarity with polymatroids.

Theoretically, standardization is determined by a matrix N with 2^n columns and $n+1$ rows such that the composite matrix $\binom{M^\sharp}{N}$ has full rank. N-standardized submodular functions are those 2^n-dimensional vectors for which both $M^\sharp r \ge 0$ and $Nr = 0$ hold. Therefore, these vectors sit in the $2^n - (n+1)$-dimensional subspace orthogonal to N. Using some alternate coordinate system of the 2^n-dimensional space with coordinates either in N or orthogonal to N, the overall dimension of the standardized functions is reduced from 2^n to $2^n - (n+1)$.

3.3. The Cone of p-Standardized Functions

In case of lower or upper standardization, the subspace $\mathcal{S}_\ell$ (or $\mathcal{S}_u$, respectively) is spanned by the lowest (topmost) $n+1$ coordinates, thus the reduction simply discards these coordinates (and replaces them by zeros). In case of p-standardization, the reduced function g is determined by the coordinates in the set

$$\mathcal{R} = \{A \subseteq X : A \neq \emptyset \text{ and } |A| \neq n{-}1\}. \tag{4}$$

Let g be a vector with coordinates in $\mathcal{R}$, and expand it to a 2^n-dimensional vector $\tilde{g}$ by defining the values at the missing places as $\tilde{g}(\emptyset) = 0$ and $\tilde{g}(A) = g(X)$ for $|A| = n-1$. Clearly, $\tilde{g}$ is in the linear space $\mathcal{S}_p$, thus g is a reduced p-standardized submodular function if and only if $M^\sharp \tilde{g} \geq 0$. This product, however, can be computed directly from g. Let M be the matrix obtained from $M^\sharp$ by deleting the column corresponding to the empty set (as $\tilde{g}$ is zero at $\emptyset$), and replacing the $n+1$ columns corresponding to subsets A with $|A| \geq n-1$ by their sum (as $\tilde{g}$ has the same value at these indices). In other words, $M = M^\sharp S_p$, where S_p is the matrix

$$S_p = \begin{array}{c} \left(\begin{array}{cc} 0 & 0 \\ \mathbf{I} & 0 \\ 0 & \mathbf{1} \end{array}\right) \end{array} \begin{array}{l} \Leftarrow 1 \text{ row} \\ \Leftarrow 2^n - (n{+}2) \text{ rows} \\ \Leftarrow n+1 \text{ rows} \end{array}$$

Here $\mathbf{I}$ is the unit matrix and $\mathbf{1}$ is a column vector where each entry is 1. Since $\tilde{g} = S_p g$, we have $Mg = M^\sharp S_p g = M^\sharp \tilde{g}$. Let us define

$$\mathcal{C} = \{g : Mg \geq 0\}, \tag{5}$$

where g is a $2^n - (n{+}1)$-dimensional vector with indices from $\mathcal{R}$. Clearly, $\mathcal{C}$ is the set of reduced and p-standardized submodular functions, therefore every submodular function on X is the sum of a modular function and the expansion of an element from $\mathcal{C}$.

Claim 4. *$\mathcal{C}$ is a full-dimensional pointed polyhedral cone.*

Proof. $\mathcal{C}$ is the intersection of $\binom{n}{2}2^{n-2}$ many half-spaces (the number or rows in M), thus it is polyhedral. All of these halfspaces contain the origin, thus $\mathcal{C}$ is the union of rays (half-lines) starting from the origin. By Claim 3, $\mathcal{C}$ is a subset of the non-negative orthant (all coordinates of a p-standardized submodular function are non-negative), thus $\mathcal{C}$ does not contain a full line; and, in particular, $\mathcal{C} \cap -\mathcal{C} = \{0\}$. To prove that $\mathcal{C}$ is full-dimensional, it is enough to show that it contains $2^n - n - 1$ many linearly independent vectors. Choose $J \subseteq X$ with at least two elements (observe that there are $2^n - n - 1$ many such subsets). For each of them, consider the function

$$f_J(A) = \begin{cases} 1 & \text{if } J \cap A \neq \emptyset, \\ 0 & \text{otherwise.} \end{cases} \tag{6}$$

It is easy to check that f_J is both submodular and p-standardized (since $|J| \geq 2$, $f_J(A) = 1$ when $|A| \geq n-1$). Linear independence of the vectors f_J can be checked directly (by induction on the number of elements in X), or by observing that their Möbius transform gives all unit vectors. For details consult Lemma 3 in [23]. □

An immediate consequence of this claim is that the matrix M in (5) has full rank.

Let us recall some further terminology of polyhedral geometry from [16,17]. A *face* of the cone $\mathcal{C}$ is a subset F of $\mathcal{C}$ with the property that if a positive convex combination of points in $\mathcal{C}$ falls into F, then the starting points are also in F. Both $\mathcal{C}$ and the empty set are faces—they are the trivial ones—, and the only single-element face is the *vertex* of the cone, which in this case is the origin. Proper faces are just the intersection of $\mathcal{C}$ and a supporting hyperplane. The dimension of a face F is its affine dimension; for pointed cones it is the maximal number of linearly independent vectors in F. One-dimensional faces are *edges* or *extremal rays*; and a *facet* is a face of codimension 1. Each proper face F is the intersection of all facets containing F. Facets of $\mathcal{C}$ lie on and are identified by the hyperplanes defined by the rows of the matrix M—this follows from point b) of Claim 2 which says that none of these rows is redundant.

Theorem 1. *Extremal submodular functions on X are unique up to a positive scaling and a shift by a modular function.*

Proof. For a submodular function f let its p-standardized and reduced image be f^o. Observe that the map $f \mapsto f^o$ is linear, and the image is the complete cone $\mathcal{C}$. Linearity implies that conic combination is preserved. Thus if f^o is not on an extremal ray of $\mathcal{C}$, then f is not extremal. At the same time, supermodular functions whose image is on some extremal ray of $\mathcal{C}$ form a generator: every other supermodular function is a positive conic combination of them. As supermodular functions whose image is the same $g \in \mathcal{C}$ differ by a modular shift only, the claim of the theorem follows. □

Theorem 2. *For a finite X there are finitely many extremal submodular functions.*

Proof. According to Theorem 1 extremal submodular functions are, up to scaling and modular shift, in a one-to-one correspondence with the extremal rays of the cone C. Every extremal ray of $\mathcal{C}$ is the intersection of the facets it is a subset of. Since $\mathcal{C}$ has $\binom{n}{2}2^{n-2}$ many facets, it has finitely many extremal rays, which proves the statement. □

Theorem 2 imposes a trivial upper bound on the number of extremal submodular functions. This is $2^{80} \approx 10^{24}$ for $n = 5$, while the actual value is around 10^5, see Table 1. For $n = 6$ we also expect a huge gap between the bound $2^{240} \approx 10^{72}$ given by this theorem and the actual value.

3.4. Symmetries

Submodular functions have many symmetries. The most notable ones are induced by permutations of the base set X. Let π be such a permutation of X, which we extend to functions on 2^X by

$$(\pi f)(A) = f(\pi A),$$

where πA is the image of $A \subseteq X$ under π. Trivially, f is submodular (supermodular or modular) iff πf is such. There is a less known symmetry of submodular functions called *reflection* [14] defined as

$$f^{\complement}(A) = f(X \setminus A),$$

arising from the permutation $A \mapsto X \setminus A$ of the subsets of X. Claim 5 essentially says that these are the only symmetries induced by permutations of subsets of X which map the complete set of inequalities in (1) onto itself.

Claim 5. *Suppose a permutation of the subsets of X induces a permutation of the inequalities in (1). Then it is equivalent to an optional reflection followed by a symmetry induced by a permutation of X.*

Proof. Let a permutation of the subsets of X be defined by the bijection φ, and rewrite the inequalities in (1) by mapping each subset A to $\varphi(A)$. For each subset, count how many times it occurs on the right-hand side of the inequalities. There will be two with the maximal count $(3^n+1)/2 - 2^n$; which means they must be $\emptyset$ and X, but also $\varphi(\emptyset)$ and $\varphi(X)$ in some order. If $\varphi(\emptyset) = X$, apply a reflection, and assume $\varphi(\emptyset) = \emptyset$ going forward. On the right hand side of the inequalities every subset occurs next to the empty set, except for singletons. This means φ is a bijection on singletons, which determines the permutation of the base set that will give rise to φ. Each pair of singletons occurs exactly once on the left hand side, with the corresponding right-hand side containing the empty set and the union of the two singletons. This means that φ for two-element subsets is uniquely determined by its values on singletons. The same logic for larger subsets yields that φ is also determined on all remaining subsets of X. □

A permutational symmetry π automatically maps to a symmetry of the cone $\mathcal{C}$. It is because π induces a permutation of the coordinate set $\mathcal{R}$ defined in (4), which, in turn, induces a permutation of the rows of the matrix M so that for each row δ of M and each vector g with coordinates in $\mathcal{R}$ we have

$$\delta \cdot g = \pi(\delta) \cdot \pi(g).$$

In particular, $Mg \geq 0$ if and only if $M\pi(g) \geq 0$, thus $\mathcal{C} = \pi(\mathcal{C})$.

The case of reflection is more subtle as $f^{\complement}$ is not necessarily p-standardized. To get the reflected pair σf of the p-standardized submodular function f one has to shift $f^{\complement}$ by the modular function

$$m(A) = -f(X) + \textstyle\sum\{f(i) : i \in A\}$$

(see part a) of Claim 1) to get the p-standardized submodular function

$$(\sigma f)(A) = f(X \setminus A) - f(X) + \textstyle\sum\{f(i) : i \in A\}. \tag{7}$$

In matroid and polymatroid parlance [24] σf is called the *dual* of f. Reflection also induces a permutation on the rows of $M^\sharp$ (and then on rows of M) as the reflected image of the elementary inequality $(i,j|K)$ is the elementary inequality $(i,j|X \setminus ijK)$. This permutation of inequalities, denoted again by σ, also satisfies the exchange property

$$\delta \cdot g = \sigma(\delta) \cdot \sigma(g).$$

Consequently $\mathcal{C} = \sigma(\mathcal{C})$, thus σ is another symmetry of the cone $\mathcal{C}$.

All of these symmetries induce a linear transformation on the coordinates, consequently they preserve extremality. Symmetric images of a fixed extremal ray form an *orbit*. As the symmetry transformations can be computed trivially, it suffices to compute one ray from each orbit. Since reflection is idempotent, there are $2n!$ many symmetries. This means a significant reduction as we expect the majority of the orbits to contain $2n!$ many different rays. Our computations confirm that this is indeed the case, see Figure 7.

3.5. Extending the Base Set

Suppose the base set X is extended by a new element y to the larger set $Y = Xy$. We extend a function f defined on the subsets of X to a function $f^\star$ defined on all subsets of Y by

$$f^\star(yA) = f^\star(A) = f(A) \ \text{ for } A \subseteq X,$$

in particular, $f^\star(y) = f(\emptyset)$. Observe that if f is a p-standardized submodular function, then so is $f^\star$. Consequently the map $g \mapsto g^\star$ embeds the cone $\mathcal{C}_X$ of reduced, p-standardized submodular functions over X into the cone $\mathcal{C}_Y$.

Claim 6. *Suppose r is an extremal ray in $\mathcal{C}_X$. Then $r^\star$ is an extremal ray in $\mathcal{C}_Y$.*

Proof. Suppose $r^\star$ is a positive conic combination of rays s_i in $\mathcal{C}_Y$, that is, $r^\star = \sum_i \lambda_i s_i$ where all λ_i are positive. We must show that s_i is a multiple of $r^\star$. Since $r^\star(y) = 0$, we have $s_i(y) = 0$. By Claim 3 the submodular function represented by s_i is monotone and pointed, thus for every $A \subseteq X$

$$0 \le s_i(Ay) - s_i(A) \le s_i(y) - s_i(\emptyset) = 0.$$

This means that there is an r_i in $\mathcal{C}_X$ such that $s_i = r_i^\star$, and then $r^\star = \sum_i \lambda_i r_i^\star$. Since $r \mapsto r^\star$ is a linear map, it implies $r = \sum_i \lambda_i r_i$. By assumption r is extremal, thus r_i is a multiple of r, and then $s_i = r_i^\star$ is a multiple of $r^\star$, as required. □

The embedding $g \mapsto g^\star$ of $\mathcal{C}_X$ into $\mathcal{C}_Y$ also preserves the symmetries of $\mathcal{C}_X$. Denote the duality symmetry on X and Y by σ_X and σ_Y, respectively. Similarly, extend the permutation π_X on X to a permutation π_Y on Y by keeping its effects on X and stipulating $\pi_Y(y) = y$.

Claim 7. *For any $g \in \mathcal{C}_X$, $(\sigma_X g)^\star = \sigma_Y g^\star$, and $(\pi_X g)^\star = \pi_Y g^\star$.*

Proof. Clearly we have $g^\star(Y) = g^\star(X) = g(X)$, and also for all subsets $A \subseteq Y$, $g^\star(Y \setminus A) = g(X \setminus A)$ and

$$\sum\{g^\star(i) : i \in A\} = \sum\{g(i) : i \in A \setminus y\}.$$

According to the above and (7), for $A \subseteq X$ we get

$$\begin{aligned}(\sigma_Y g^\star)(A) &= (\sigma_Y g^\star)(Ay) = \\ &= g(X \setminus A) + \sum\{g(i) : i \in A\} = (\sigma_X g)(A),\end{aligned}$$

as was claimed. The second statement can be checked analogously. □

According to Claim 6, extremal submodular functions on an $n+1$-element base contain the extremal submodular functions on a base with n elements—and then, by induction, all extremal submodular functions on smaller bases. Claim 7 strengthens this result: if, instead, we consider orbits, then every orbit on a smaller base occurs exactly once as an orbit on a larger base. For example, taking those orbits for $n = 6$ which have a representative vanishing on some singleton, we get exactly the 672 orbits of $n = 5$, see Table 1.

4. Methods for Generating Extremal Rays

This section gives an overview of some methods from polyhedral geometry [17,25] which were used in our computations. Recall that the problem of finding all extremal submodular functions was reduced to the task of finding all extremal rays of a pointed, full-dimensional, polyhedral cone. For this exposition the dimension of the cone is denoted by d, and the cone is the intersection of $m \ge d$ irredundant positive halfspaces specified as

$$\mathcal{C} = \{x \in \mathbb{R}^d : Mx \ge 0\},$$

where the matrix M has d columns, m rows, and has rank d. Irredundant means that no proper subset of the rows of M define the same cone. In other words, the hyperplanes determined by the rows of M are the bounding facets (that is, $d-1$-dimensional faces) of $\mathcal{C}$.

The *ray* generated by a non-zero point (or vector) $x \in \mathbb{R}^d$ is $r = \{\lambda x : \lambda \ge 0\}$. By abusing the notation a ray is identified with one (or any) of its generating points. Thus we write $r \in \mathcal{C}$ to mean that all points of r are in $\mathcal{C}$, and $Mr \ge 0$ to mean that this relation holds for some non-zero (and then all) points of r.

The ray r is *extremal* in $\mathcal{C}$ if it is a one-dimensional face of $\mathcal{C}$. Recalling the definition of polyhedral faces [16], r is extremal iff it is not a strictly positive conic combination of other rays of $\mathcal{C}$.

The *support* of the ray $r \in \mathcal{C}$, denoted by $M[r]$, is the set of those bounding hyperplanes of $\mathcal{C}$ the ray is on, namely $M[r] = \{a \in M : a \cdot r = 0\}$. Worded differently, $M[r]$ is the set of active constraints for r, or the set of facets of $\mathcal{C}$ containing r. The following claim follows easily from the definitions, see also [16,17].

Claim 8. *The ray $r \in \mathcal{C}$ is extremal if and only if $M[r]$ has rank $d-1$, and then r is the unique one-dimensional solution of the homogeneous system $(M[r])\,x = 0$.*

The number of facets r is on is called its *weight* and is denoted by $w(r)$; this number is called *incidence number* in [26]. Clearly, $w(r)$ is the number of rows in $M[r]$. If $M[r]$ has rank $d-1$, then it must have at least $d-1$ rows. Consequently extremal rays have weight $w(r) \geq d-1$.

4.1. The Double Description Method

A polyhedral cone is determined both by its bounding hyperplanes arranged into the matrix M, and by the set of its extremal rays arranged into the matrix R. Several algorithms and implementations exist for enumerating the rows of R given the matrix M, see [25,27]. Interestingly, the converse problem, namely enumerating the bounding hyperplanes given the set of extremal rays, is a completely equivalent problem. This is because if the rows of R are interpreted as hyperplanes, then the cone they generate has exactly M as the set of its extremal rays, see, e.g., [16,17].

The ray enumeration algorithm that fits the highly degenerate case of submodular functions best is the iterative *Double Description Method*, abbreviated as DD. It was described first by Motzkin et al. [28]; for a recent overview see [29]. DD is a variant of the Fourier-Motzkin elimination, or Chernikova's algorithm.

As an illustration, we describe how DD enumerates the vertices of a d-dimensional polytope given by its facets. The method starts by selecting $d+1$ of the provided facets which form a simplex. Each of its $d+1$ vertices lies on exactly d facets. Computing their coordinates requires solving $d+1$ linear systems of equations with d unknowns.

An iterative step starts with an intermediate polytope bounded by $d+i$, $i \geq 1$, of the given facets such that all vertices of this polytope are known. This polytope is then cut by one of the remaining facets as illustrated in Figure 1, yielding the polytope of the next step. The vertices of the next polytope are computed as follows. First, the vertices of the old polytope are separated into three groups: those which are on the positive side of the cutting facet, those which are on that facet (the zero group), and those which are on its negative side. Vertices in the positive and in the zero group will be vertices of the new polytope as well. All additional vertices are on the cutting facet, and can be obtained as the intersection of the cutting facet and an old edge which connects a positive and a negative vertex. DD stops when there are no more facets to be processed, and then it provides the required vertex list.

The iterative step requires recognizing whether two vertices of the intermediate polytope form an *edge*. Analogs of the tests described in Claim 9 are used by almost all DD packages.

The DD variant which is used to enumerate all extremal rays of the cone determined by the matrix M is sketched as Code 1.

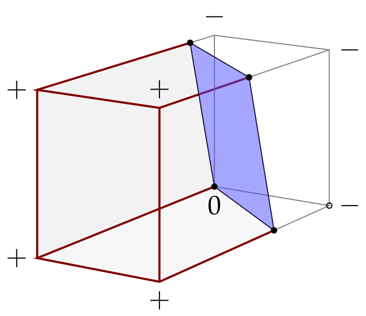

Figure 1. An intermediate step of the double description method. The polytope from the previous iteration is cut by a new facet. The new vertices are at the intersection of the new facet and an old edge with a positive and a negative endpoint. The deleted part is shown with gray edges.

It starts with a subset M_0 of d inequalities from M which defines a full-dimensional cone with exactly d extremal rays. In an intermediate step we have a cone determined by the inequalities (or facets) in $M_i \subseteq M$, and also we have the complete list R_i of its extremal rays. This way each intermediate cone has a double description, giving the name of the method. To obtain the next cone, a new inequality a is added to M_i, that is, part of the cone is cut off by the hyperplane corresponding to $a \geq 0$. This hyperplane partitions the old rays in R_i into positive, negative, and zero ones depending on whether the ray is on the positive side of the hyperplane, on the negative side, or it is on the hyperplane. Positive and zero rays remain extremal in the next cone; negative rays are part of the removed segment, and so are not part of the new cone. All additional extremal rays are on the cutting hyperplane. These are the rays where the conic span of an *adjacent pair* of a positive and a negative ray intersects this hyperplane; for details, see Section 8.1 in [17]. The algorithm terminates when there are no more inequalities to add.

Code 1: The Double Description Method

```
1  Compute the initial DD pair $(M_0, R_0)$; $i \leftarrow 0$
2  **while** there is $a \in M \setminus M_i$ **do**
3      $M_{i+1} \leftarrow M_i \cup \{a\}$
4      Split $R_i$ into positive/negative/zero rays
5      $R_{i+1} \leftarrow$ positive and zero rays
6      **for each** $r_1$ positive/$r_2$ negative ray **do**
7          **if** $r_1$ and $r_2$ are adjacent **then**
8              Compute the ray $r = \text{conic}(r_1, r_2) \cap a$
9              $R_{i+1} \leftarrow R_{i+1} \cup \{r\}$
10         **end**
11     **end**
12     $i \leftarrow i + 1$
13 **end**
```

The induction step (lines 3 to 11 of the outlined DD algorithm) was implemented as a stand-alone program working as a *pipe*. It reads the DD pair (M_i, R_i) together with the new inequality $a \in M$, and produces the next DD pair (M_{i+1}, R_{i+1}) which is amenable for the next iteration. The initial DD pair (line 1) is created by the controlling program using simple linear algebraic tools [19]. The overhead caused by the pipe arrangement (shipping the data in and out of memory at each iteration) is more than compensated for by simpler data structure, smaller memory requirement, and the ability to allocate all necessary memory in a single request (thus avoiding memory fragmentation). As a bonus, almost linear speed-up can be achieved by distributing the computation made by the pipe program among several cores or, preferably, over several machines. Figure 8 gives an illustration of the general performance by plotting the total number of rays against the speed of ray generation. The figure depicts data for the equivalent problem of enumerating adjacent rays, see Section 4.2.

The crucial part of the DD algorithm is testing the adjacency of extremal rays of the current cone (line 7 in Code 1). This ensures that only extremal rays of the next iteration are added to the output. The following two equivalent criteria for ray adjacency are from [29].

Claim 9 (Fukuda–Prodon [29])**.** *For two extremal rays r_1 and r_2 the following conditions are equivalent.*

(a) *r_1 and r_2 are adjacent;*
(b) *the rank of $M[r_1] \cap M[r_2]$ is $d-2$ (algebraic test);*
(c) *there is no extremal ray r apart form r_1 and r_2 such that $M[r] \supseteq M[r_1] \cap M[r_2]$ (combinatorial test).*

If we write $w(r_1, r_2)$ to mean the number of rows in $M[r_1] \cap M[r_2]$, then the algebraic test requires computing the rank of a matrix with d columns and $w(r_1, r_2)$ rows. The naive implementation requires $O(d^2 w(r_1, r_2))$ arithmetic operations, independently of the number of extremal rays. As this computation is sensitive to numerical errors, the rank is frequently computed using arbitrary precision arithmetic, which can be very slow even for small values of d. Conversely, the combinatorial test runs in time proportional to the number of extremal rays, potentially a very large number. However, this test can be implemented using fast bit operations on the ray–facet incidence matrix, and is not prone to numerical errors. Both tests can be sped up using a simple consequence of the algebraic test: r_1 and r_2 are definitely *not* adjacent if $w(r_1, r_2)$ is smaller than $d-2$. This simple condition should be checked before delving into any of the two more demanding tests.

4.2. Enumerating Neighbors

Another avenue to finding extremal rays of a cone is to find extremal rays adjacent to a known extremal ray [26,30–32], also known as *Adjacency Decomposition*. Since extremal rays of a cone are connected with respect to the adjacency relation (Theorem 3.14 in [16]), starting from any extremal ray of the cone and determining its neighbors, then the neighbors of these rays, and so on, will eventually generate all extremal rays.

For the details let r be such a fixed extremal ray of the cone $\mathcal{C} = \{x \in \mathbb{R}^d : Mx \geq 0\}$. Enumerating the neighbors of r can be done by solving a generic ray enumeration problem in dimension $d-1$. To show that this is the case, let $M' = M[r]$ be the set of facets r is on, and let $z \in M \smallsetminus M'$ be any of the remaining facets of $\mathcal{C}$. Clearly $M'r = 0$ and $z \cdot r > 0$ as z is a bounding facet of $\mathcal{C}$, and therefore r is on the positive side of z. Consider the following $(d-1)$-dimensional cone $\mathcal{C}'$ embedded into the d-dimensional space $\mathbb{R}^d$:

$$\mathcal{C}' = \{x \in \mathbb{R}^d : M'x \geq 0 \text{ and } z \cdot x = 0\}.$$

Since r is extremal, M' has rank $d-1$. Since $M'r = 0$ and $z \cdot r \neq 0$, z is not in the linear span of the rows of M', therefore the composite matrix $\binom{M'}{z}$ has rank d, and then $\mathcal{C}'$ is $(d-1)$-dimensional, as was claimed. The analog of Claim 8 remains valid: a ray $s \in \mathcal{C}'$ is extremal if and only if $M'[s]$ has rank $d-2$. Similarly, both tests for adjacency of extremal rays stated in Claim 9 remain true for $\mathcal{C}'$ if $d-2$ is replaced by $d-3$. Therefore the Double Description Method with some minor modifications can be used to enumerate the extremal rays of the cone $\mathcal{C}'$.

Claim 10. *Extremal rays of $\mathcal{C}'$ and the neighboring rays of r are in a one-to-one correspondence.*

Proof. We begin by describing the mapping from rays adjacent to r to extremal rays in $\mathcal{C}'$. Let r_1 be an extremal ray of $\mathcal{C}$ adjacent to r. Let $s = r_1 - \lambda r$ where λ is chosen so that $z \cdot s = 0$. (Since $z \cdot r \neq 0$, such a λ exists.) We show that s is an extremal ray of $\mathcal{C}'$. First, $s \in \mathcal{C}'$ as $z \cdot s = 0$ and

$$M's = M' \cdot (r_1 - \lambda r) = M' r_1 \geq 0$$

as r_1 is a ray in $\mathcal{C}$. Second, $M'[s] = M'[r_1] = M[r_1] \cap M[r]$. Since r and r_1 are adjacent, the rank of this matrix is $d-2$, which proves that s is extremal by Claim 9. It is easy to see that no two neighbors of r are mapped to the same ray of $\mathcal{C}'$.

In the other direction, let $s \in \mathcal{C}'$ be extremal, and consider the the ray $r' = s + \mu r$ for some real number μ. Clearly, $M'[r'] = M'[s]$, and this matrix has rank $d-2$. If a is a row in M', then $a \cdot r' = a \cdot s \geq 0$. If a is in M but not in M', then $a \cdot r$ is positive, and

$$a \cdot r' = a \cdot (s + \mu r) = a \cdot s + \mu(a \cdot r) \geq 0$$

if and only if $\mu \geq -(a \cdot s)/(a \cdot r)$. Choosing μ as the minimum of these values, r' will be a ray in $\mathcal{C}$. Moreover, $M[r']$ will contain the row $a \notin M'$ where this minimum is taken: $M[r'] \supseteq M'[r'] \cup \{a\}$. Since $M'[r']$ has rank $d-2$ (r' is extremal in $\mathcal{C}'$), $M[r']$ has rank $d-1$, proving that r' is extremal in C. Finally, $M[r] \cap M[r'] = M'[r']$ has rank $d-2$, thus r and r' are adjacent by Claim 9. This construction also indicates how neighbors of r can be generated from the extremal rays of $\mathcal{C}'$. □

The same reduction can be applied iteratively to the reduced cone $\mathcal{C}'$. Suppose, in general, that a k-dimensional cone $\mathcal{C}$ embedded in the d-dimensional space is defined as

$$\mathcal{C} = \{ x \in \mathbb{R}^d : Mx \geq 0, \text{ and } Ax = 0 \}$$

where M has m rows, d columns, and rank k; A has $d-k$ rows, d columns, and rank $d-k$; and the composite matrix $\binom{M}{A}$ has rank d. Let r be an extremal ray of $\mathcal{C}$. Enumerating the neighbors of r can be reduced to enumerating the extremal rays of the $k-1$-dimensional cone $\mathcal{C}'$ defined by the matrix pair $M[r]$ and $\binom{A}{a}$ where the row $a \in M$ is not in $M[r]$.

The reduction stops when the complexity of the cone C defined by the matrix pair (M, A) becomes manageable thus its extremal rays can be enumerated directly. For this purpose one can use the d-dimensional DD method directly with some minor modifications as discussed above. Another possibility is to make $\mathcal{C}$ full dimensional first by projecting it to the k-dimensional space

$$\pi\mathcal{C} = \{ y \in \mathbb{R}^k : (MP)y \geq 0 \}$$

where P is a generator of the nullspace of A (see [19] for details). Extremal rays of $\mathcal{C}$ can be recovered by applying P to the extremal rays of $\pi\mathcal{C}$. Actually, such a projection was used for the p-standardized submodular functions in Section 3.3.

4.3. The Kernel Method

When applying the DD method directly to enumerate all extremal rays of the cone

$$\mathcal{C} = \{ x \in \mathbb{R}^d : Mx \geq 0, \text{ and } Ax = 0 \},$$

computations simplify considerably when each row of M contains exactly one non-zero entry. Using homogeneity and an optional coordinate sign change, that entry can be assumed to be 1. When the DD method adds the new row $a \in M$, old rays are put into positive, negative, and zero parts depending on the sign of the inner product $a \cdot r$, see line 4 of Code 1. Similarly, inner products $a \cdot r_1$ and $a \cdot r_2$ are used in line 8 when computing that conic combination of rays r_1 and r_2 which lies on the hyperplane determined by the row a. If the only non-zero coordinate in a is $a[i] = 1$, then computing these inner products reduces to the trivial task of taking the i-th coordinate. Additionally, ray coordinates where the corresponding row in M contains zeros only need not to be stored at all; and among the processed coordinates (determined by those rows of M which were previously handled by DD) it suffices to store only one bit indicating whether that coordinate of the ray is zero or not. When DD finishes, these bits determine the matrix $M[r]$, which allows (re)computing the coordinates of the ray r, postponing a large part of high-precision computations to the final stage.

This variant of the DD method is called *kernel* or *null-space method* [33]. While the kernel method uses very specially defined cones, it is universal, as the standard ray enumeration problem for the cone

$$\mathcal{C} = \{ x \in \mathbb{R}^d : Mx \geq 0 \}$$

can be transformed into the required special form in the following way. Suppose M has m rows. Let I_m be the $m \times m$ unit matrix and $y \in \mathbb{R}^m$ be new *slack* variables. The first d coordinates of the extremal rays of the cone

$$\{ (x, y) \in \mathbb{R}^{d+m} : y \geq 0, Mx + I_m y = 0 \}$$

are just the extremal rays of $\mathcal{C}$. The software package POLCO uses the kernel method for ray enumeration. For a detailed description and theoretical background of the package, see [33].

5. Improving the Combinatorial Test

In the DD method we need to check the adjacency of each pair of extremal rays taken from the two sides of the new facet, that is, where one ray comes from the positive class, and the other ray from the negative; see line 7 in Code 1. The majority of these pairs are expected not to be adjacent, in which case the pair is skipped, thus this test needs to be as fast as possible. This Section discusses the details and potential improvements of the combinatorial test described in Claim 9.

Let M be the set of facets (inequalities), and R be the set of extremal rays of the cone $\mathcal{C}$. The ray–facet *incidence matrix* has one row for each facet and one column for each ray, and for $a \in M$ and $r \in R$ the entry at position (a, r) is 1 if the scalar product $a \cdot r$ is zero (the ray r lies on the facet a, they are incident), and 0 otherwise. For each $a \in M$ let $\widehat{a}$ be the 0–1 string formed from the entries in row a of this incidence matrix, and, similarly, $\widehat{r}$ be the 0–1 string formed from the entries in column r. Clearly, $M[r]$ contains those rows of M where $\widehat{r}$ is 1. Denoting the number of 1's in $\widehat{r}$ by $|\widehat{r}|$, we have $|\widehat{r}| \geq d-1$ for every $r \in R$. This is because by Claim 8 $M[r]$ has rank $d-1$, thus it has at least $d-1$ rows. Similarly, the number of rows in both $M[r_1]$ and $M[r_2]$, which was denoted by $w(r_1, r_2)$ above, is $|\widehat{r}_1 \cap \widehat{r}_2|$, where the intersection of two strings is understood to be taking the minimal value at each position.

Code 2 outlines the combinatorial test. Line 1 executes the quick precheck $w(r_1, r_2) \geq d-2$ coming from the algebraic test; it filters out many of the cases. Line 2 initializes the bitstring b, and the loop in lines 3–6 computes those bits in b whose index r satisfies $M[r] \supseteq M[r_1] \cap M[r_2]$ with the exception of r_1 and r_2 whose indices are cleared in the initialization. Finally, r_1 and r_2 are adjacent if no other ray remains in b, that is, b becomes the all zero string. This condition is checked in the loop, skipping the remaining computation whenever possible.

Code 2: Combinatorial adjacency test of rays r_1 and r_2.

```
1  if $|\widehat{r}_1 \cap \widehat{r}_2| < d-2$ then return $no$
2  $b \leftarrow$ all 1 bitstring of length $|R|$;
       set $b[r_1] \leftarrow 0$ and $b[r_2] \leftarrow 0$
3  for each $a \in \widehat{r}_1 \cap \widehat{r}_2$ do
4      $b \leftarrow b \cap \widehat{a}$
5      if $b = 0$ then return $yes$
6  end
7  return $no$
```

N. Zolotykh made the observation [34] that if the ray r violates the combinatorial test, that is,

$$M[r] \supseteq M[r_1] \cap M[r_2],$$

then both $M[r] \cap M[r_1]$ and $M[r] \cap M[r_2]$ contain the intersection $M[r_1] \cap M[r_2]$ as a subset. In particular, if $|\widehat{r}_1 \cap \widehat{r}_2| \geq d-2$ (this pair survived the quick test), then both $|\widehat{r} \cap \widehat{r}_1| \geq d-2$ and $|\widehat{r} \cap \widehat{r}_2| \geq d-2$. Thus it suffices to restrict the search in the loop at 3–6 of Code 2 to such rays. So for a ray r_i let $\widehat{g}(r_i)$ (for *graph test*, a terminology used in [34]) be the bitstring indexed by the rays such that at index r this string has 1 if r and r_i differ and $|\widehat{r} \cap \widehat{r}_i| \geq d-2$, and has 0 otherwise. Instead of line 2 in Code 2, b can be initialized to $b \leftarrow \widehat{g}(r_1) \cap \widehat{g}(r_2)$. Since b starts with fewer bits set, the loop at 3–6 is expected to finish earlier.

Computing the strings $\widehat{g}(r_1)$ and $\widehat{g}(r_2)$ for all positive and negative rays takes time, and it is not clear that this overhead is compensated for by the improved performance. The main drawback, however, is the significant memory needed to store these bit strings. The best compromise to retain some of the advantages of the graph test without extensive memory requirement seems to be computing $\widehat{g}(r_1)$ "on the fly" for positive rays, also suggested in [34]. Our contribution is to complement this by not using $\widehat{g}(r_2)$ for negative rays at all. Performing the adjacency tests in an appropriate order every $\widehat{g}(r_1)$ needs to be computed only once and the string may occupy the same memory location. This version, dubbed as *½-graph test*, is sketched as Code 3.

The loop in lines 2–4 prepares the bitstring $\widehat{g}(r_1)$; this will be done only once for every positive ray r_1 under an appropriate scheduling. The quick check in line 6 reuses part of this precomputation: the result of the condition $|\widehat{r}_1 \cap \widehat{r}_2| < d-2$ is simply looked up in the bitstring $\widehat{g}(r_1)$. Line 7 initializes b and the loop at lines 8–11 searches for a ray r such that $M[r]$ extends $M[r_1] \cap M[r_2]$. The search is done in (typically 64 bit) chunks rather than over the whole string at once. When a chunk becomes empty, (that is, all zero), which we expect to happen frequently, the inner loop is aborted, further improving the performance.

Code 3: ½-graph adjacency test of rays r_1 and r_2,

```
1  if $\widehat{g}(r_1)$ is not defined then
2      for each $r \in R$ do
3          $\widehat{g}(r_1)[r] \leftarrow (r \neq r_1$ and $|\widehat{r} \cap \widehat{r}_1| \geq d-2)$
4      end
5  end
6  if $\widehat{g}(r_1)[r_2] = 0$ then return $no$
7  $b \leftarrow \widehat{g}(r_1)$; set $b[r_2] \leftarrow 0$
8  for each $a \in \widehat{r}_1 \cap \widehat{r}_2$ do
9      $b \leftarrow b \cap \widehat{a}$
10     if $b = 0$ then return $yes$
11 end
12 return $no$
```

6. Enumerating Extremal Submodular Functions for $n = 6$

We are now ready to apply the tools and algorithms described above to specific values of n. To recap, Section 3.3 defined the cone of p-standardized submodular functions over a base set with $n \ge 3$ elements. Extremal submodular functions, up to a shift by a modular function, were identified with the extremal rays of a polyhedral cone $\mathcal{C}_n$. Enumerating extremal submodular functions means enumerating these extremal rays. The cone $\mathcal{C}_n$ sits in the $d = 2^n - (n+1)$-dimensional space and is defined by $m = \binom{n}{2}2^{n-2}$ homogeneous inequalities as

$$\mathcal{C}_n = \{ g \in \mathbb{R}^d : M_n\, g \ge 0 \},$$

see Section 3.3. Rows of M_n are determined by the p-standardized elementary inequalities as discussed in Section 3.1. Each non-zero entry in M_n is either $+1$ or -1, and each row contains two, three or four non-zero entries only, making M_n to be extremely sparse. As shown in Section 3.4, $\mathcal{C}_n$ has $2n!$ symmetries. The $n!$ part comes from permuting the base set, and the factor 2 comes from reflection. Symmetric images of extremal rays are also extremal, thus it suffices to have one representative from each orbit, that is, symmetry class.

Based on the above numbers and on later calculations, Table 1 lists the dimension, number of inequalities, symmetries, total number of extremal rays, number of orbits, and the weight range (see the comment after Claim 8) of cones for different values of n.

Table 1. Dimension, facet number, symmetries, extremal rays, orbits, and weight range.

n	d	m	Symm	Rays	Orbits	Weight
3	4	6	12	5	2	3–4
4	11	24	48	37	7	10–20
5	26	80	240	117,978	672	25–72
6	57	240	1440	$>3.9{\cdot}10^{11}$	$>2.6{\cdot}10^{8}$	56–225

Section 3.5 defines a canonical embedding of $\mathcal{C}_n$ into $\mathcal{C}_{n+1}$; this embedding preserves both extremal rays (by Claim 6), and orbits (by Claim 7). In particular, extremal rays of $\mathcal{C}_n$ can be recovered from the extremal rays of $\mathcal{C}_{n+1}$ by simply looking for rays which take zero at a coordinate labeled by a fixed singleton. Referring to Table 1, preserving the orbits means that among the 672 orbits of $\mathcal{C}_5$ there are seven which are the images of the seven orbits of $\mathcal{C}_4$. Similarly, among all orbits of $\mathcal{C}_6$ there are exactly 672 in which some (or equivalently, every) ray takes zero at some singleton, and exactly seven orbits in which the rays take zero at two or more singletons.

6.1. Weight (Incidence) Distribution

The weight distribution provides information about how degenerate the cone is. A d-dimensional polyhedral cone is *simple*, or *non-degenerate*, if every extremal ray has the minimal possible weight $d - 1$. Figure 2 depicts the weight distribution of the extremal rays of $\mathcal{C}_5$. The distribution has a long tail of rays with large weights. The estimated weight distribution for $\mathcal{C}_6$ on Figure 3 reveals a similar picture.

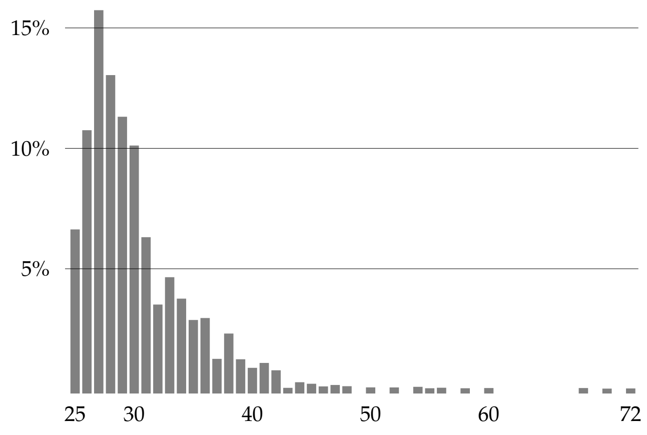

Figure 2. Weight distribution for $n = 5$.

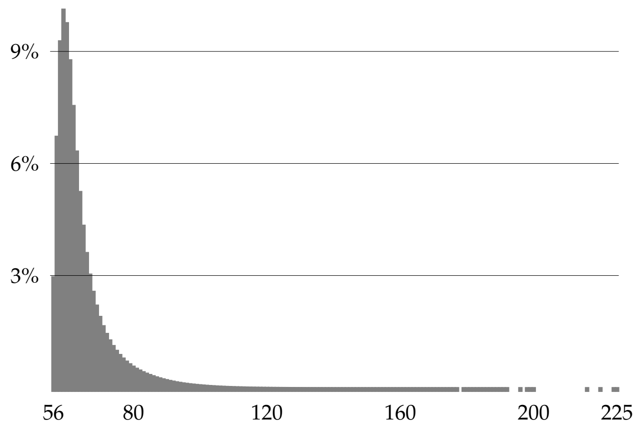

Figure 3. Estimated weight distribution for $n = 6$.

To trace the rays with large weights, consider the submodular function f_J defined in (6) for $J \subseteq X$, $|J| \geq 2$. This function is the rank function of a connected matroid [24], and the rank function of every connected matroid is extremal [35]. Consequently, f_J is an extremal ray of $\mathcal{C}_n$. These rays (and their duals) have large weights. Denoting $|J|$ by $k \geq 2$, f_J satisfies all but $\binom{k}{2}2^{n-k}$ of the elementary inequalities: the exceptional ones are the triplets $(i, j|K)$ where $i, j \in J$ and K is disjoint from J. This means that f_J has weight

$$\binom{n}{2}2^{n-2} - \binom{k}{2}2^{n-k}.$$

For $n = 3, 4, 5$ the extremal rays with largest weight are the rays f_J where $|J| = 2$ (and their symmetric versions); the corresponding weights are 4, 20 and 72, respectively, barely below the maximal possible weight values of 5, 23, and 79, which is one less than the number of facets. For $n \geq 6$, however, the largest weight is attained when $|J| = n$, that is, by f_X. The weight of f_X is $\binom{n}{2}(2^{n-2} - 1)$, which exceeds the cone dimension by a factor of $n^2/8$.

The weight distributions depicted on Figures 2 and 3 clearly show that the degeneracy of the submodular cones is caused not by a few rays with large weights, but by the fact that almost all possible weights occur across the complete weight range.

6.2. Insertion Order

As discussed in Section 4, conventional wisdom dictates to use the Double Description method when the cone has a high degree of degeneracy, see also [18] and Section 3 in [36]. The DD method, however, is highly sensitive to the insertion order of the facets. Without careful ordering, the number of extremal rays in an intermediate cone can be exponentially larger than the final number of rays [17,18,29], rendering the DD method unusable. Insertion strategies are categorized by the way they are applied: *static orderings* are determined and applied in advance and the ordering is fixed during the computation, while *dynamic orderings* allow for determining the order of the remaining inequalities dynamically, depending on the state of the computation.

The static ordering strategy called *lex-min* sorts the rows of the matrix lexicographically, and applies them in increasing order. An example for a dynamic strategy is *max-cut* (resp. *min-cut*), which chooses the unprocessed matrix row which cuts off the maximal (resp. minimal) number of extremal rays from the actual intermediate cone. Experimental assessment of different strategies indicated, see [27,29], that any one can work reasonably well for some enumeration problem, but could fail badly for others. The overall recommendation is to use lex-min or some of its refinements, as this strategy consistently outperforms all others, frequently by orders of magnitude. See Chapter 8.1 in [17] or Chapter 3.2 in [33].

In the specific case of submodular cones, however, better insertion strategies may exist, especially as the lex-min ordering is sensitive to the order of the coordinates, while the DD method itself is not. Moreover, previous experience showed that existing ordering strategies lead to an "overshoot" of the intermediate cones in terms of the number of extremal rays, before this number is reduced to the final result. This is illustrated by the size of the last few intermediate cones, depicted on Figure 4, where the DD method was applied to $\mathcal{C}_5$ using the lex-min inserting strategy with several random permutations of the coordinates. Our aim was to find an insertion strategy that avoids this phenomenon.

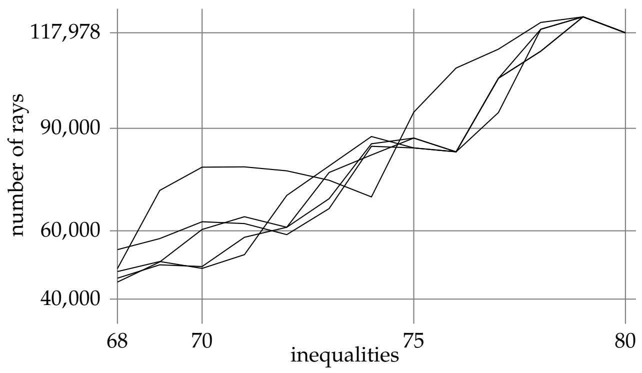

Figure 4. Size of intermediate cones for the lex-min inserting order using randomly shuffled coordinates.

The method to achieve this goal was a backward greedy algorithm, which calculated the insertion order in *reverse*. We first determined which facet (which one out of the m rows of the matrix M) should be added *last* so that the penultimate cone would have the smallest possible number of extremal rays. Fortunately, this task requires solving at most m ray enumeration problems, as the penultimate cone is independent of the insertion order of its $m-1$ facets. Next, further elements of the insertion order were determined similarly, always making sure that the initial sequence of inequalities had full rank.

We determined the insertion order experimentally for $\mathcal{C}_5$, and aimed to generalize the resulting order to $n = 6$. To illustrate the choice for the last inequality, note from Table 1 that $\mathcal{C}_5$ has dimension $d = 26$ and is bounded by $m = 80$ facets. Also, the facets of $\mathcal{C}_5$, or, equivalently, the rows of the defining matrix M_5, correspond to the elementary inequalities $(i, j|K)$. Denote the elements of the 5-element base set X by digits from 0 to 4. Taking permutational equivalence into account there are only four different elementary inequalities, namely

$$(0,1|\emptyset), \ (0,1|2), \ (0,1|23), \text{ and } (0,1|234).$$

As discussed in Section 3.4, reflection (or duality) maps $(0,1|\emptyset)$ to $(0,1|234)$, and maps $(0,1|2)$ to $(0,1|34)$, and the latter one is permutationally equivalent to $(0,1|23)$. Consequently only two cases has to be considered for the last position: it is either $(0,1|\emptyset)$ or $(0,1|2)$. In the first case the remaining inequalities determine the penultimate cone with 112,712 extremal rays, in the second case that cone has 122,642 extremal rays. Therefore $(0,1|\emptyset)$ (or one of its equivalents) should be inserted last.

We abstracted the patterns discovered in the experimental results into the *t-opt* (for "tail-optimal") *insertion order* defined below. Note that the experimental results do not determine the order of the first d inequalities.

For the definition of the t-opt insertion order, fix an ordering of the n-element base set X. A subset K of X is identified with the string of length $|K|$ enlisting elements of K in increasing order; the elementary inequality $(i,j|K)$ is always written so that i precedes j, similarly to the examples above.

Definition 1 (T-opt insertion order)**.** *Two elementary inequalities are in the relation* $(i_1,j_1|K_1) \prec (i_2,j_2|K_2)$ *iff*

- *when the subsets* K_1 *and* K_2 *have different number of elements then* $|K_1|$ *precedes* $|K_2|$ *in the list* $0, n-2, 1, n-3, 2, n-4, \ldots$
- *when* K_1 *and* K_2 *have the same number of elements then the string* $i_1j_1K_1$ *is lexicographically smaller than the string* $i_2j_2K_2$*.*

The t-opt insertion order *is the reverse of the order* $\prec$*.*

For the base set $X = \{0,1,2,3\}$ the order $\prec$ is illustrated by

$$(2,3|\emptyset) \prec (1,2|03) \prec (0,1|3) \prec (1,3|2);$$

moreover, $(0,1|\emptyset)$ is the smallest, and $(2,3|1)$ is the largest one in this ordering.

The blue curve on Figure 5 depicts, on logarithmic scale, the size of the intermediate cones when the t-opt insertion order is used to enumerate the extremal rays of $\mathcal{C}_5$. The curve increases steadily (as opposed to the cases where the lex-min order was used), and seems to be approximately linear, indicating a steady exponential growth. Using this insertion order for $\mathcal{C}_5$ the total execution time of our implementation of the DD method running on a single core of an Intel i5-4590 CPU was under 1 min, beating all other tested insertion strategies. For comparison, the same task on the same single core using POLCO v.4.7.1 [33] took 1.9 min, and using *lrs* (lrslib v.7.3, [37]) took 590 min.

When the t-opt insertion order was applied to $\mathcal{C}_6$, the number of extremal rays in the intermediate cones grew steadily, as expected. After completing 44 iterations this number reached 18,506,227. At this point the computation was stopped as the last iteration took over 200 h to finish, and the next iteration was estimated to require five to ten times as much running time, see Section 6.4. The size of intermediate cones up to this point is plotted as the blue curve on Figure 6. Assuming it is similar to the $n=5$ case, this initial segment gives an estimate for the total number of extremal rays of $\mathcal{C}_6$ somewhere between 10^{20} and 10^{30}.

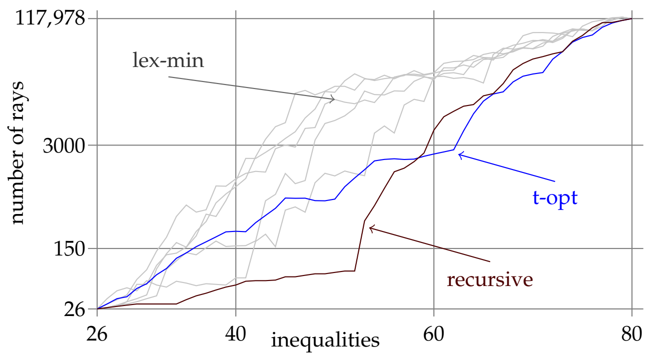

Figure 5. Size of intermediate cones for t-opt (blue), recursive (red), and lex-min (gray) insertion orders for $n = 5$.

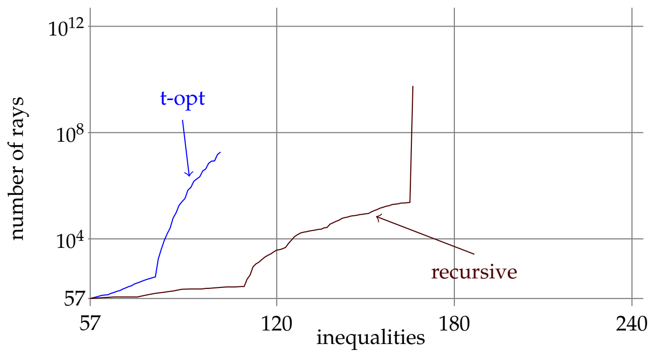

Figure 6. Size of intermediate cones for t-opt (blue) and recursive (red) insertion orders for $n = 6$.

Due to the excessive time the next iteration would have taken, the DD algorithm for $\mathcal{C}_6$ and using the t-opt insertion order had to be stopped quite early. Therefore we searched for a different inserting regime which would allow processing more inequalities in reasonable time. It was observed that when $K_1 \subset K_2 \subset K_3 \subset \cdots$ inserting the inequalities

$$(i,j|K_1),\ \ (i,j|K_2),\ \ (i,j|K_3),\ \ \ldots$$

in this order resulted in a moderate increase in the number of the extremal rays. This observation motivated our definition of the *recursive order*. For the definition we assume again that the elements of X are ordered. The recursive procedure $\mathsf{SUB}(K)$ enumerates the subsets of $K \subseteq X$, where K is specified as an increasing list of its elements. The enumeration is defined as follows:

- List K itself first;
- For all i in K in decreasing order, call $\mathsf{SUB}(K \smallsetminus i)$ and keep the first occurrence of each emitted subset.

In particular, $\mathsf{SUB}(0,1,2,3)$ lists $\{0,1,2,3\}$ first, then calls $\mathsf{SUB}(0,1,2)$, $\mathsf{SUB}(0,1,3)$, $\mathsf{SUB}(0,2,3)$ and $\mathsf{SUB}(1,2,3)$. Also, $\mathsf{SUB}(0,1)$ produces $\{0,1\},\{0\},\emptyset,\{1\}$ in this order, and $\mathsf{SUB}(0,1,2)$ produces

$$\{0,1,2\},\{0,1\},\{0\},\emptyset,\{1\},\{0,2\},\{2\},\{1,2\}.$$

Definition 2 (Recursive insertion order)**.** *Enumerate the elementary inequalities as follows. First, take the two-element subsets ij of X with $i < j$ in lexicographic order. For each such pair ij, enumerate all subsets K of $X \smallsetminus ij$ by calling $\mathsf{SUB}(X \smallsetminus ij)$ and append $(i,j|K)$ to the list.*

The recursive insertion order *is the reverse of this enumeration.*

As an example, for $X = \{0,1,2,3\}$ the recursive enumeration of the inequalities starts with $(0,1|23)$, $(0,1|2)$, $(0,1|\emptyset)$, $(0,1|3)$, $(0,2|13)$, and ends with $(2,3|0)$, $(2,3|\emptyset)$, $(2,3|1)$. The insertion order is the reverse: it starts with $(2,3|1)$ and ends with $(0,1|23)$.

Figures 5 and 6 also show the performance of the recursive insertion order. Overall, it is slightly worse than the t-opt order (using about 15% more time to generate all extremal rays of $\mathcal{C}_5$), but it allows inserting significantly more inequalities before the size of the intermediate cone suddenly increases. In case of $\mathcal{C}_6$, as shown by Figure 6, the last intermediate cone before the size jump has 165 facets out of the 240, and has only 235,961 extremal rays. In the next iteration the number of extremal rays jumps to 5,733,451,485.

Undoubtedly both insertion strategies are variants of lex-min. Nevertheless, our experiments showed that the recursive order has some intricate connection with the structure of the submodular cone. Denoting the smallest element of the n-element base set X by 0, recall from Definition 2 that the elementary inequalities $(0,j|K)$ form an end segment of the recursive order. The intermediate cone $\mathcal{C}_n^*$ defined by inserting all other inequalities has the moderate number of rays

$$|\mathcal{C}_n^*| = 2|\mathcal{C}_{n-1}| + (n-1). \tag{8}$$

For example, $|\mathcal{C}_5^*| = 78$ as $|\mathcal{C}_4| = 37$ from Table 1, while the total number of rays of $\mathcal{C}_5$ is 117,978. Similarly, $|\mathcal{C}_6^*| = 235{,}961 = 2|\mathcal{C}_5| + 5$, which is negligible compared to the estimated number of rays of $\mathcal{C}_6$. Equation (8) follows from the fact that the bounding inequalities

$$\{\, (i,j|K),\ (i,j|0K) : i,j \in X \smallsetminus 0 \text{ and } K \subseteq X \smallsetminus 0ij \,\}$$

of $\mathcal{C}_n^*$ define two disjoint copies of $\mathcal{C}_{n-1}$ on two disjoint subsets of the coordinates (that is, subsets of X), namely those which do not contain the element 0, and those which do. The additional $(n-1)$ term in (8) comes from the fact that the first copy is not p-standardized, see Section 3.3. The recursive nature of this insertion strategy implies that similar reductions occur earlier, explaining the recursive pattern of the initial segments of the red curves in Figures 5 and 6.

Next, consider the step in the DD method after—or applied to—$\mathcal{C}_n^*$. It exhibits an intriguing structural property of the submodular cone, which explains the sudden jump in the number of extremal rays in the next iteration.

Claim 11. *Suppose the DD method cuts the cone $\mathcal{C}_n^*$ by the inequality $(0,i|K)$. Among the extremal rays of $\mathcal{C}_n^*$ there will be exactly one more positive than negative rays. Moreover, all positive/negative ray pairs are adjacent.*

Proof. As was done above, split the coordinates of $\mathcal{C}_n$ (that is, subsets of X) into two classes depending on whether they contain the minimal element 0. Put coordinates containing 0 first, followed by the others. An extremal ray of $\mathcal{C}_n^*$ has zeros exclusively at the coordinates in the first part, or zeros exclusively in the second part. Furthermore, the non-all-zero part is an extremal ray of $\mathcal{C}_{n-1}$. Therefore, extremal rays of $\mathcal{C}_n^*$ are the concatenations of a zero and a non-zero vector in either $\langle r,\mathbf{0}\rangle$ or $\langle \mathbf{0},r\rangle$ order where r is an extremal ray of $\mathcal{C}_{n-1}$. It is easy to check that the algebraic condition in Claim 9 for adjacency holds for all "opposite" ray pairs $\langle r_1,\mathbf{0}\rangle$ and $\langle \mathbf{0},r_2\rangle$, thus all these pairs are actually adjacent extremal rays of $\mathcal{C}_n^*$, see also [16]. Choosing the inequality $(0,i|K)$ as the next one to be inserted by the DD method, the extremal rays of $\mathcal{C}_n^*$ are split into positive, zero, and negative ones depending on which side of the hyperplane corresponding to $\delta(0,i|K)$ they are on. Observe that for every extremal ray r of $\mathcal{C}_{n-1}$ we have

$$0 \le \delta(0,i|K)\cdot\langle r,\mathbf{0}\rangle = -\delta(0,i|K)\cdot\langle \mathbf{0},r\rangle.$$

Consequently, either both $\langle r, \mathbf{0} \rangle$ and $\langle \mathbf{0}, r \rangle$ are in the zero part, or the first one is in the positive, and the second one is in the negative part—meaning that the positive and negative parts have equal size. Moreover, each positive / negative ray pair is adjacent, and so they produce a new ray in the next iteration. We also need to account for the additional $n-1$ extremal rays in the first copy of $\mathcal{C}_{n-1}$ in $\mathcal{C}_n^*$, of which exactly one goes into the positive part, and the other $(n-2)$ go into the zero part, proving the first statement of the claim. The same reasoning as above shows that this single additional ray in the positive part is adjacent to all the negative ones, proving the second claim. $\square$

In summary, independently which of the $(0, i|K)$ inequalities is added next to $\mathcal{C}_n^*$, the number of positive rays will be one more than the number of negative rays, and each positive/negative ray pair will define an extremal ray in the next iteration. Specifically for $n = 6$, the next inequality with which the recursive order intersects $\mathcal{C}_6^*$ is $(0, 5|34)$. Among the 235,961 extremal rays of $\mathcal{C}_6^*$ the number of positive, zero, negative ones relative to this inequality are 75,719; 84,524; and 75,718, respectively. Thus the number of extremal rays in the next iteration is

$$75{,}719 + 84{,}524 + 75{,}719 \times 75{,}718 = 5{,}733{,}451{,}485,$$

as was claimed earlier. A ray of this cone has 57 coordinates. Using only two bytes to store a coordinate value, the list of extremal rays of this iteration would occupy more than 650 gigabytes.

6.3. Generating the First Extremal Rays of $\mathcal{C}_6$

Extremal rays of $\mathcal{C}_5$, as discussed in Section 3.5, automatically provide extremal rays of $\mathcal{C}_6$ via the extension $r(A) = r(Ay) = r'(A)$ where r' is extremal in $\mathcal{C}_5$.

Our first attempt to generate further extremal rays was to use Claim 8. Rows of the generating matrix M_6 were chosen randomly until the chosen rows formed a submatrix of rank $d-1$. The one-dimensional solution r of this homogeneous system was then checked against $M_6 r \ge 0$, that is, whether $r \in \mathcal{C}_6$. If yes, then r provided an extremal ray of $\mathcal{C}_6$. While every extremal ray had a chance to be found, mostly rays with extremely large weights (see Figure 3) were generated, due to the high number of submatrices generating the same ray. Running even for a considerable time, this method generated only a few thousand *essentially different* rays, that is, rays from different orbits, see Section 3.4. The efficiency improved only marginally when we tried to restrict the search to the neighborhood of a ray r that was found earlier. It was done by fixing a $d-1$-row submatrix of M_6 which determines this r. When choosing rows of M_6 which were to generate the next ray, the first $d-2$ rows were chosen from this submatrix. By Claim 9 extremal rays generated this way are those which are adjacent to r.

Another possibility to generate extremal rays of $\mathcal{C}_6$ is indicated by the following observation. If r is an extremal ray of any intermediate subcone of the DD method so that r is also an element of $\mathcal{C}_6$, then r is an extremal ray of $\mathcal{C}_6$. It is because every extremal ray of an intermediate cone is either an extremal ray of the next iteration, or it is outside of the next cone, and thus of $\mathcal{C}_6$.

We ran the DD algorithm for several steps, and checked which extremal rays of the last iteration were actually elements of $\mathcal{C}_6$. Since by Claim 3 extremal rays of $\mathcal{C}_6$ have non-negative coordinates, it was worth checking if $r \ge 0$ before delving into the more time consuming computation of $r \in \mathcal{C}_6$. Furthermore, the check $r \in \mathcal{C}_6$ can be incorporated into the inner loop of the lastly executed DD step. Code 4 details the replacement of the loop at lines 6–11 of Code 1. It postpones the expensive adjacency test for those pairs which would otherwise produce an extremal ray of $\mathcal{C}_6$.

Line 3 is the quick precheck of the combinatorial test detailed in Code 2. If r_1 and r_2 pass this test, then the potential new ray is computed and checked for being an element of $\mathcal{C}_6$. The adjacency test is performed only if this is the case, resulting in a significant speed-up. This modified DD iteration was used on top of the intermediate cones produced by the t-opt insertion order using different choices for the next inequality.

Code 4: Modified inner loop in the last round of DD

```
1   – previous cone is to be split by the row $a \in M_6$
2   for each $r_1$ positive/$r_2$ negative ray do
3       if $|\widehat{r}_1 \cap \widehat{r}_2| < d-2$ then continue
4       compute the ray $r = \text{conic}(r_1, r_2) \cap a$
5       if not $r \geq 0$ then continue
6       if not $M_6 r \geq 0$ then continue
7       if $r_1$ and $r_2$ are adjacent then
8           report $r$ as extremal in $\mathcal{C}_6$
9       end
10  end
```

Starting from $\mathcal{C}_6^*$ as the intermediate cone, by Claim 11 every positive/negative ray pair of this cone is adjacent. Therefore the checks in lines 3 and 7 of Code 4 should not be executed at all when using the modified iteration on top of $\mathcal{C}_6^*$.

Overall, these processes provided about a half million essentially different extremal rays of $\mathcal{C}_6$, that is, representatives of that many different orbits.

6.4. Visiting the Neighborhood

Adjacency decomposition is a natural approach when the underlying problem has many symmetries, see [30–32]. As described in Section 4.2, adjacency decomposition starts with some initial extremal rays of $\mathcal{C}$, generates their neighbors, then the neighbors of these neighbors, etc., until no new ray can be generated. Adjacency decomposition requires solving several enumeration problems in one less dimension, typically with a much smaller number of constraints. It can enumerate all extremal rays even in cases when the original DD method would exhaust all available resources [26]. The efficiency is partially due to the fact that adjacency decomposition can take advantage of cone symmetries, while the DD method cannot. It is so as it suffices to find the neighbors of a single representative from each orbit (that is, symmetry class), as neighbors of rays from the same orbit are the symmetrical images of neighbors of this representative. Adjacency decomposition can also be applied recursively to the most difficult subproblems. Typically those difficult subproblems have a large number of symmetries as well, improving the efficiency further.

A clear resource limit of the DD method is the total number of the extremal rays to be enumerated. The same limit for the adjacency decomposition method is the number of orbits. According to Figure 7, in case of the submodular cone $\mathcal{C}_6$, each orbit contains, with minimal exceptions, $2n! = 1440$ rays. Consequently adjacency decomposition could reduce the complexity of enumerating extremal rays by three orders of magnitude.

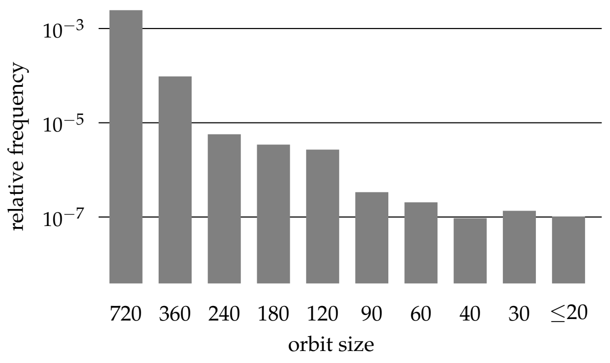

Figure 7. Relative frequency of orbits of $\mathcal{C}_6$ with a size below 1440.

Enumerating neighbors should be started with low-weight rays as in these "easy" cases the DD method can be used directly as outlined in Section 4.2. This approach was successfully used to generate 260 M essentially different extremal rays of $\mathcal{C}_6$ (that is, rays on different orbits), starting from the rays computed earlier. This required computing neighbors of only about 450,000 rays. These rays had typically low weights: 69% had weights between 56 and 59, 21% had weights 60–69, and the remaining 10% had weights between 70 and 80. On average, each probed ray produced over 600 neighbors in different orbits. More than 95% of these computations took less than 1 s.

Figure 8 depicts a sample of longer computations showing the total number of generated neighboring rays versus the speed of generation on the same CPU and using a single core. The computation used either the t-opt order (blue dots) or the recursive order (red dots), see Section 6.2, restricted to the rows of the corresponding submatrix. While the recursive order seems to appear mainly in the slower region, in some cases it was significantly faster than the t-opt order. The plotted data seem to follow, alas with large deviation, a linear trend marked by the green line. Since both coordinates are on logarithmic scale, it gives the exponential approximation

$$\text{speed} = C \cdot \text{size}^{-0.6}$$

for some constant C. Thus if the output size increases tenfold, then the speed goes down by a factor of $10^{-0.6} \approx 0.25$, and the total running time is expected to go up by a factor of 40.

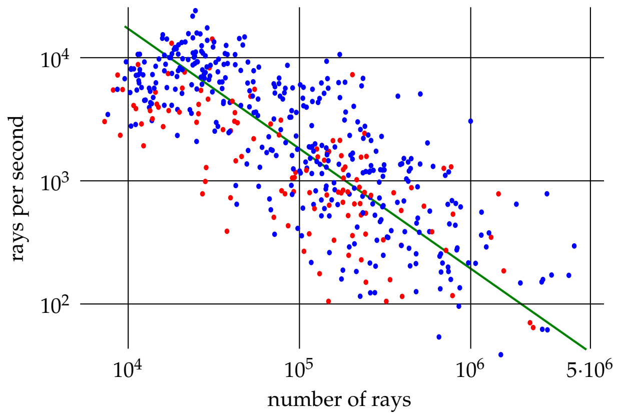

Figure 8. Speed of generating adjacent rays. Blue: t-opt order; red: recursive order.

The weight distribution for rays and for orbits are almost identical as only a small fraction of the orbits is not maximal (see Figure 7). Using the data of Figure 3, 90% of the weights are expected to be below 75, and so belong to the "easy" cases. There are, however, "difficult" cases as well. As discussed in Section 4, almost all weights occur between the smallest and the largest possible values with example of rays with large weights being the extremal rays f_J defined in (6). Table 2 lists how many neighboring orbits the rays f_J have. For $n = 5$ the exact numbers are shown in parentheses, for $n = 6$ the percentages are estimates. The data implies that even listing the neighboring orbits of these extremal rays requires space comparable to the total number of orbits. For $n = 6$ finding all neighbors of f_{01}, or at least checking whether all of them have been found, would require efforts comparable to enumerating all orbits of the extremal rays of $\mathcal{C}_6$.

Table 2. Number of neighboring orbits of the heaviest extremal rays for $n = 5$ (top) and $n = 6$ (bottom).

$\lvert J \rvert$	$w(f_J)$	**Orbits**
2	72	100% (672)
3	68	99% (664)
4	68	95% (636)
5	70	44% (299)
$\lvert J \rvert$	$w(f_J)$	**Orbits**
2	224	89%
3	216	83%
4	216	89%
5	220	76%
6	225	37%

7. Estimating the Total Number of Rays and Orbits for $n = 6$

While we successfully generated a large number of extremal rays of $\mathcal{C}_6$, generating a complete list with any of the above methods would require an unrealistic amount of resources. We still aimed to provide an estimate for their overall number to understand the expected complexity of the complete problem. However, to the best of our knowledge, no efficient randomized algorithm exists for computing a reasonable approximation of the total number of extremal rays of a polyhedral cone, and only two general approaches have been proposed. The first method, developed by Avis & Devroye [38], is based on the backtrack tree size estimator of Knuth, and was implemented around the reverse search (RS) vertex enumeration algorithm [39]. This estimator, while theoretically unbiased, has enormous variance and in many cases vastly underestimates the number of rays. The implementation in the software package *lrs* [37] gave the estimates ≈12,000 for $\mathcal{C}_5$ and ≈102,000 for $\mathcal{C}_6$ consistently. (The parameter *maxdepth* was set to 2 and the iteration count was set to 100.) The fact that $\mathcal{C}_5$ has almost ten times as many rays renders these estimates too inaccurate to be useful.

The second estimation method from [40] is based on MCMC (Markov Chain Monte Carlo) and uses conditional sampling to reduce the variance. The algorithm, however, assumes that choosing $(d-1)$ rows of the generating matrix M randomly samples its maximal $(d-1)$-rank subsets uniformly. This is not the case for the highly degenerate submodular cones. In order to be able to use this estimation method, an effective sampler of the maximal, $(d-1)$-rank subsets of M would have to be devised first. This task seems to be an equally difficult and challenging problem. With such a sampler our first attempt to create extremal rays of $\mathcal{C}_6$ as described in Section 6.3 would have been significantly more efficient.

To get an estimate on the total number of extremal rays of $\mathcal{C}_6$ we resort to a heuristic argument without theoretical guarantee. From the existing pool of 260M essentially different rays, 1000 were chosen randomly with the following restrictions: the ray was not used in the pool creation, and the ray weight is at most 70. According to Figure 3 these restrictions exclude about 15% of rays. All neighbors of the chosen 1000 rays were computed, producing 2,824,119 extremal rays. These rays determined 2,797,684 distinct orbits (99%), of which 154,170 (about 5.5%) have already had a representative in the pool. Assuming that the pool is a completely random subset of all orbits, this yields the estimate 260M / 0.055 $\approx$ 4.7·10^9 for the total number of orbits. From here, based on the weight distribution depicted on Figure 3, the number of extremal rays is estimated to be 6.5·10^{12}. Repeating this experiment gave similar results.

This estimate is clearly biased as the pool is far from random: it was created by adding neighbors, in several stages, to a relatively small number of rays. While the above figures are quite reasonable, we expect the actual count to be significantly larger, closer to the lower estimate 10^{20} obtained in Section 6.2.

8. Conclusions

Generating extremal submodular functions on a finite base set is an intriguing and challenging problem, as a list of these functions carries invaluable information about their structure. Knowledge of all extremal function for the case $|X| = 4$ was instrumental in investigating properties of conditional independence structures [8,9]. Extremal submodular functions for $|X| = 5$ were used to investigate and create new non-Shannon type entropy inequalities [10], and to characterize conditional independence structures [11]. Generating this list for the base set $|X| = 6$ was shown in our paper to be equivalent to enumerating the extremal rays of a $d = 57$-dimensional polyhedral cone determined by $m = 240$ facets, see Table 1. The main difficulty stems from the high dimensionality, leading to both structural and numerical problems. For ray enumeration we used a variant of the *Double Description* method (Code 1). The iteration step was implemented as a *pipe*: as input it gets the double description of an intermediate cone and the cutting facet, and it outputs the double description of the cone for the next iteration. During an iteration ray adjacency is checked by the ½-*graph* combinatorial test, see Code 3. This test uses some pre-computation to speed up the traditional combinatorial test of Fukuda and Prodon (Claim 9), while keeping the required additional memory manageable.

The *insertion order* is the order in which the rows of the matrix M determining the facets of the cone are processed by DD. This order has a huge impact on performance, and without careful ordering the size of the intermediate cones can be exponentially larger than the final one. We defined two orders which performed excellently for smaller base sizes: the *t-opt* order in Definition 1, and the *recursive* order in Definition 2. Both orders provided a steadily increasing intermediate cone size for the control case $|X| = 5$. For the $|X| = 6$ case the t-opt order enabled performing 44 DD-iterations with the last cone having over 18M extremal rays. The last iteration took over 200 h to complete on the specified hardware. In contrast, the recursive order let us proceed much further, performing 101 iterations. Using this ordering, the next iteration, which was *not* computed, would produce a cone with more than 5700 M extremal rays.

At this point it was clear that the DD method is not powerful enough to generate all extremal submodular functions, so we looked for alternative solutions. First, we proved that extremal functions on a smaller base set are also extremal on a larger base set, see Claim 6. This automatically provides many extremal functions for $|X| = 6$. Second, if r is such an extremal ray of an intermediate cone in the DD method so that r is inside the final cone (that is, it satisfies all defining inequalities), then r is also extremal in the final cone, see the discussion in Section 6.3. Using a modified DD iteration (Code 4), these extremal submodular functions can be extracted more efficiently than generating the next iteration and then checking for inclusion in the final cone.

The third option, which turned out to be the most successful, is the *Adjacency Decomposition* discussed in Section 6.4. Given an extremal ray of the final cone, we can enumerate all of its *neighbors*. This method also leverages the symmetry of the problem as it suffices to find the neighbors of a single ray from the orbit of its symmetrical versions. Since a typical orbit has the maximal 1440 elements (see Figure 7), the gain is significant. The DD enumeration with certain modifications was used for this purpose, see Section 4.2.

Generating neighbors repeatedly allowed us to generate 260 M extremal functions, all on different orbits. This provided a significant sample sample size which was used for estimating different properties, such as weight distribution (Figure 3), frequency of different orbit sizes (Figure 7), or how many orbits are represented by the neighbors of the heaviest extremal rays (Table 2). Using our dataset, we attempted to estimate the total number of extremal rays and orbits, for details, see Section 7.

The recursive insertion order revealed an intriguing structural property of the submodular cone, see Section 6.2. To recap this property, the cone $\mathcal{C}_n^*$, which is determined by the bounding inequalities

$$\{\,(i,j|K),\ (i,j|0K) : i,j \in X \smallsetminus 0 \text{ and } K \subseteq X \smallsetminus 0ij\,\}$$

is the direct product of two instances of $\mathcal{C}_{n-1}$ on two disjoint subsets of the coordinates. Consequently, extremal rays of $\mathcal{C}_n^*$ can be generated directly from the extremal rays of $\mathcal{C}_{n-1}$ *without any computation*. The cone determined by the remaining inequalities

$$\{(0,j|K) : j \in X \smallsetminus 0 \text{ and } K \subseteq X \smallsetminus j\,\}$$

is quite simple. Apart from its linearity space, extremal rays are

$$r_A : J \mapsto \begin{cases} 0 & \text{if } J \supseteq A, \\ 1 & \text{otherwise,} \end{cases}$$

for all $A \subseteq X \smallsetminus 0$. It is an open question whether this observation can be used to speed up the overall computation.

Section 3.4 discussed the symmetries of the submodular functions. In Claim 5 we proved that this set has no additional symmetry which would be induced by a permutation of the defining inequalities. Nevertheless, the submodular cone may have additional symmetries. It is an open problem to find additional symmetries, or to prove that there are no more.

High-dimensional geometrical problems are prone to numerical instability, a well-known phenomenon in computational geometry. Determining the rank of a sparse 57-dimensional matrix with ± 1 non-zero entries requires high-precision arithmetic. The coordinates of the final (and also the intermediate) extremal rays are solutions of such a homogeneous linear system of equations with 57 variables, thus these numbers can be scaled to be integers. Their magnitude can theoretically be as high as 10^{10}. Quite surprisingly, none of the coefficients in our computation had an absolute value above 1800, in spite of the fact that we explicitly tried to generate rays with large coefficients. Future research could reveal a theoretical explanation for this surprising fact.

We have made the dataset of the generated extremal rays available to allow and welcome future research to build on it and add more extremal rays, either by using our algorithms, or by any other method developed independently. If the total number of functions is our lowest estimate, then the complete dataset would contain around 1000 times as many rays as it contains now, occupying around 50 terabytes.

Author Contributions: Conceptualization, E.P.C. and L.C.; Methodology, E.P.C. and L.C.; Software, E.P.C. and L.C.; Writing—original draft, E.P.C. and L.C. All authors have read and agreed to the published version of the manuscript.

Funding: The research reported in this paper was partially supported by the ERC Advanced Grant ERMiD.

Data Availability Statement: Implementations of the main algorithms discussed in this paper can be found at https://github.com/csirmaz/submodular-functions-6, accessed on 18 December 2024. Partial results of the computations representing 360 billion extremal submodular functions in 260 M orbits are available as a Zenodo dataset at https://zenodo.org/records/13954788, accessed on 19 October 2024.

Acknowledgments: We would like to thank the careful work of the reviewers, which helped us to improve the presentation.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Balcan, M.F.; Harvey, N.J. Submodular functions: Learnability, structure, and optimization. *SIAM J. Comput.* **2018**, *47*, 703–754. [CrossRef]
2. Bach, F. Learning with Submodular Functions: A Convex Optimization Perspective. In *Foundations and Trends in Machine Learning*; Now Publishers: Hanover, MA, USA, 2013; Volume 6.
3. Studený, M.; Bouckaert, R.R.; Kocka, T. *Extreme Supermodular Set Functions over Five Variables*; Technical Report 1977; Institute of Information Theory and Automation: Prague, Czech Republic, 2000.
4. Tao, T. *An Introduction to Measure Theory*; Graduate Studies in Mathematics, American Mathematical Society: Providence, RI, USA, 2021.
5. Sadeghi, O. The Diminishing Returns (DR) Property and Its Applications in Machine Learning. Ph.D. Thesis, University of Washington, Electrical and Computer Engineering, Seattle, WA, USA, 2023.
6. Kurzweil, R. The Law of Accelerating Returns. In *Alan Turing: Life and Legacy of a Great Thinker*; Teuscher, C., Ed.; Springer: Berlin/Heidelberg, Germany, 2004; pp. 381–416. [CrossRef]
7. Kashimura, T.; Sei, T.; Takemura, A.; Tanaka, K. Cones of Elementary Imsets and Supermodular Functions: A Review and Some New Results. In *Harmony of Gröbner Bases and the Modern Industrial Society*; Hibi, T., Ed.; World Scientific Publishing Company: Singapore, 2012; pp. 117–152. [CrossRef]
8. Matúš, F.; Studený, M. Conditional Independences among Four Random Variables 1. *Comb. Probab. Comput.* **1995**, *4*, 269–278. [CrossRef]
9. Studený, M. Conditional Independence Structures Over Four Discrete Random Variables Revisited: Conditional Ingleton Inequalities. *IEEE Trans. Inf. Theory* **2021**, *67*, 7030–7049. [CrossRef]
10. Csirmaz, L. One-adhesive polymatroids. *Kybernetika* **2020**, *56*, 886–902. [CrossRef]
11. Boege, T.; Bolt, J.; Studený, M. Self-adhesivity in lattices of abstract conditional independence models. *Discret. Appl. Math.* **2025**, *361*, 196–225. [CrossRef]
12. Heckerman, D. A Tutorial on Learning with Bayesian Networks. In *Innovations in Bayesian Networks: Theory and Applications*; Holmes, D., Jain, L., Eds.; Springer: Berlin/Heidelberg, Germany, 2008; pp. 33–82. [CrossRef]
13. Scutari, M. Entropy and the Kullback–Leibler Divergence for Bayesian Networks: Computational Complexity and Efficient Implementation. *Algorithms* **2024**, *17*, 24. [CrossRef]
14. Studený, M. *Probabilistic Conditional Independence Structures*; Springer: Berlin/Heidelberg, Germany, 2010.
15. Kaya, İ.C.; Ulus, F. An iterative vertex enumeration method for objective space based vector optimization algorithms. *RAIRO-Oper. Res.* **2021**, *55*, S2471–S2485. [CrossRef]
16. Ziegler, G.M. *Lectures on Polytopes*; Number 152 in Graduate texts in mathematics; Springer: Berlin/Heidelberg, Germany, 1995.
17. Fukuda, K. *Polyhedral Computation*; Department of Mathematics, Institute of Theoretical Computer Science ETH Zurich: Zürich, Switzerland, 2020. [CrossRef]
18. Avis, D.; Jordan, C. Comparative computational results for some vertex and facet enumeration codes. *arXiv* **2016**, arXiv:1510.02545.
19. Strang, G. *Introduction to Linear Algebra*, 6th ed.; Cambridge University Press: Cambridge, UK, 2023.
20. Yeung, R. *A First Course in Information Theory*; Information Technology: Transmission, Processing and Storage; Springer: Berlin/Heidelberg, Germany, 2012.
21. Matúš, F.; Csirmaz, L. Entropy Region and Convolution. *IEEE Trans. Inf. Theory* **2016**, *62*, 6007–6018. [CrossRef]
22. Studený, M. Basic facts concerning supermodular functions. *arXiv* **2016**, arXiv:1612.06599.
23. Matúš, F. Two Constructions on Limits of Entropy Functions. *IEEE Trans. Inf. Theor.* **2007**, *53*, 320–330. [CrossRef]
24. Oxley, J.G. *Matroid Theory (Oxford Graduate Texts in Mathematics)*; Oxford University Press, Inc.: Oxford, UK, 2006.

25. Joswig, M.; Theobald, T. *Polyhedral and Algebraic Methods in Computational Geometry*; Universitext; Springer: Berlin/Heidelberg, Germany, 2013.
26. Sikirić, M.D.; Schuermann, A.; Vallentin, F. Classification of eight dimensional perfect forms. *Electron. Res. Announc. Am. Math. Soc.* **2007**, *13*, 21–32. [CrossRef]
27. Avis, D.; Bremner, D. How good are convex hull algorithms? In Proceedings of the Eleventh Annual Symposium on Computational Geometry, SCG '95, New York, NY, USA, 5–7 June 1995; pp. 20–28. [CrossRef]
28. Motzkin, T.S.; Raiffa, H.; Thompson, G.L.; Thrall, R.M. The Double Description Method. In *Contributions to the Theory of Games*; Kuhn, H.W., Tucker, A.W., Eds.; Chapter 3; Princeton University Press: Princeton, NJ, USA, 1953; Volume II, pp. 51–74. [CrossRef]
29. Fukuda, K.; Prodon, A. Double description method revisited. In Proceedings of the Combinatorics and Computer Science, Hong Kong, China, 17–19 June 1996; Deza, M., Euler, R., Manoussakis, I., Eds.; Springer: Berlin/Heidelberg, Germany, 1996; pp. 91–111.
30. Bremner, D.; Sikirić, M.D.; Schuermann, A. Polyhedral representation conversion up to symmetries. *Polyhedral Computation*; ETH Zurich: Zürich, Switzerland, 2009; pp. 45–71. [CrossRef]
31. Martínez, J.M.M.; Urías, J. An algorithm for constructing the skeleton graph of degenerate systems of linear inequalities. *PLoS ONE* **2017**, *12*, e0175819. [CrossRef]
32. Rehn, T. Polyhedral Description Conversion up to Symmetries. Ph.D. Thesis, Otto von Guericke University: Magdeburg, Germany, 2010.
33. Terzer, M. Large Scale Methods to Enumerate Extreme Rays and Elementary Modes. Doctoral Thesis, ETH Zurich, Zürich, Switzerland, 2009. [CrossRef]
34. Zolotykh, N.; Bastrakov, S. Two variations of graph test in double description method. *Comput. Appl. Math.* **2019**, *38*, 100. [CrossRef]
35. Nguyen, H.Q. Semimodular Functions and Combinatorial Geometries. *Trans. Am. Math. Soc.* **1978**, *238*, 355–383. [CrossRef]
36. Assarf, B.; Gawrilow, E.; Herr, K.; Joswig, M.; Lorenz, B.; Paffenholz, A.; Rehn, T. Computing convex hulls and counting integer points with polymake. *Math. Program. Comput.* **2017**, *9*, 1–38. [CrossRef]
37. Avis, D. A Revised Implementation of the Reverse Search Vertex Enumeration Algorithm. In *Polytopes—Combinatorics and Computation*; Kalai, G., Ziegler, G.M., Eds.; Birkhäuser: Basel, Switzerland, 2000; pp. 177–198. [CrossRef]
38. Avis, D.; Devroye, L. Estimating the number of vertices of a polyhedron. *Inf. Process. Lett.* **2000**, *73*, 137–143. [CrossRef]
39. Avis, D.; Fukuda, K. A pivoting algorithm for convex hulls and vertex enumeration of arrangements and polyhedra. *Discret. Comput. Geom.* **1992**, *8*, 295–313. [CrossRef]
40. Salomone, R.; Vaisman, R.; Kroese, D.P. Estimating the number of vertices in convex polytopes. In Proceedings of the International Conference on Operations Research and Statistics. Global Science and Technology Forum, Singapore, 18–19 January 2016; pp. 96–105.