Centre de Recherches Mathématiques
CRM Proceedings and Lecture Notes
Volume **48**, 2009

Generating Vertices of Polyhedra and Related Problems of Monotone Generation

Endre Boros, Khaled Elbassioni, Vladimir Gurvich, and Kazuhisa Makino

Abstract. The well-known vertex enumeration problem calls for generating all vertices of a polyhedron, given by its description as a system of linear inequalities. Recently, a number of combinatorial techniques have been developed and applied successfully to a large number of monotone generation problems in different areas. We consider four such techniques and give examples where they are applicable to vertex enumeration. We also discuss their limitations and sketch an NP-hardness proof for generating the vertices of general polyhedra.

1. Introduction

1.1. Vertex enumeration. The well-known Minkowski–Weyl theorem states that any convex polyhedron $P \subseteq \mathbb{R}^n$ can be represented in the following two ways.

- (*H-representation*) the intersection of finitely many halfspaces:

$$P = \{x \in \mathbb{R}^n : Ax \le b\}, \tag{1.1}$$

  where $A \in \mathbb{R}^{m\times n}$ is an $m \times n$-real matrix and $b \in \mathbb{R}^m$ is an m-dimensional real vector.
- (*V-representation*) the Minkowski sum of the convex hull of a set of vectors and the conic hull of a set of directions in $\mathbb{R}^n$:

$$P = \operatorname{conv}\{v_1, \dots, v_r\} + \operatorname{cone}\{d_1, \dots, d_s\}, \tag{1.2}$$

2000 *Mathematics Subject Classification.* Primary 52B55; Secondary 68Q25.

Key words and phrases. Polytope, polyhedron, vertex, enumeration problem, monotone generation, graph, directed graph, path, cut, hypergraph, transversal, dualization, system of linear inequalities, integer programming.

This research was partially supported by DIMACS, a collaborative project of Rutgers University, Princeton University, AT&T Labs-Research, Bell Labs, NEC Laboratories America and Telcordia Technologies, as well as affiliate members Avaya Labs, HP Labs, IBM Research, Microsoft Research, Stevens Institute of Technology, Georgia Institute of Technology and Renselaer Polytechnic Institute. DIMACS was founded as an NSF Science and Technology Center.

This is the final form of the paper.

where $\mathcal{V}(P) = \{v_1, \dots, v_r\} \subseteq \mathbb{R}^n$ is the set of *vertices* or *extreme points* of P, $\mathcal{D}(P) = \{d_1, \dots, d_s\} \subseteq \mathbb{R}^n$ is the set of *extreme directions* of P, and

$$\operatorname{conv}\{v_1, \dots, v_r\} = \left\{ \sum_{i=1}^{r} \lambda_i v_i : \sum_{i=1}^{r} \lambda_i = 1, \lambda_1 \geq 0, \dots, \lambda_r \geq 0 \right\},$$
$$\operatorname{cone}\{d_1, \dots, d_s\} = \left\{ \sum_{i=1}^{s} \mu_i d_i : \mu_1 \geq 0, \dots, \mu_s \geq 0 \right\},$$

see, e.g., [48, 54] for a good introduction to polyhedral theory. In this article, we assume that all vectors and matrices are rational, and denote by L the bit size of A and b. For simplicity, we assume furthermore that the polyhedron P is *pointed*, i.e., has no line and hence has at least one vertex. We assume familiarity with standard terminology of graphs, networks and hypergraphs (see, e.g., [4, 48]).

The *vertex enumeration problem* calls for generating all the vertices $\mathcal{V}(P)$ of a polyhedron P, given by its H-representation. As we shall see, the problem can be formulated as a monotone generation problem. This motivates us to consider a number of combinatorial techniques that have been recently applied to problems of this type in different areas. We focus on four such techniques and give examples of vertex enumeration problems when they become applicable, and also discuss some of their limitation. In the appendix, we present some techniques to prove hardness of generation problems, and apply them to show that the vertex enumeration problem for polyhedra is NP-hard.

Clearly, the size of the vertex set $\mathcal{V}(P)$ can be (and typically is) exponential in n or m, and thus when we consider the computational complexity of the vertex enumeration problem, one can only hope for *output-sensitive* algorithms, i.e., those whose running time depends not only on n, m and L, but also on $|\mathcal{V}(P)|$. In particular, we consider the following decision and generation problems:

$\mathrm{Dec}(P, \mathcal{X})$**:** Given a polyhedron P, represented by a system of linear inequalities (1.1), and a subset $\mathcal{X} \subseteq \mathcal{V}(P)$ of its vertices, is $\mathcal{X} = \mathcal{V}(P)$?

$\mathrm{Inc\,Gen}(P, \mathcal{X})$**:** Given a polyhedron P, represented by a system of linear inequalities (1.1), and a subset of its vertices $\mathcal{X} \subseteq \mathcal{V}(P)$, find a new vertex in $\mathcal{V}(P) \setminus \mathcal{X}$, or indicate that there is none.

$\mathrm{Gen}(P)$**:** Given a polyhedron P, represented by a system of linear inequalities (1.1), generate all elements of $\mathcal{V}(P)$.

When the polyhedron P is bounded, the decision problem is sometimes referred to as the *Polytope-Polyhedron problem* [40]. One can distinguish different notions of efficiency, according to the time/space complexity of the generation problem:

- *Output polynomial* or *Total polynomial*: Problem $\mathrm{Gen}(P)$can be solved in $\mathrm{poly}(n, m, L, |\mathcal{V}(P)|)$ time.
- *Incremental polynomial*: Problem $\mathrm{Inc\,Gen}(P, \mathcal{X})$ can be solved in $\mathrm{poly}(n, m, L, |\mathcal{X}|)$ time, for every $\mathcal{X} \subseteq \mathcal{V}(P)$.
- *Polynomial delay*: Problem $\mathrm{Inc\,Gen}(P, \mathcal{X})$ can be solved in $\mathrm{poly}(n, m, L)$ time. In other words, the time required to generate a new element of $\mathcal{V}(P)$ is polynomial only in the input size.
- *Polynomial space*: The total space required to solve $\mathrm{Gen}(P)$ is bounded by a $\mathrm{poly}(n, m, L)$. This is only possible if the algorithm looks at no more than $\mathrm{poly}(n, m, L)$ many outputs that it has already generated.

- *Strongly P-enumerable*: $\mathcal{V}(P)$ can be enumerated with amortized polynomial delay (i.e., in $\mathrm{poly}(n,m,L)|\mathcal{V}(P)|$ time) using polynomial space.
- *NP-hard*: the decision problem $\mathrm{Dec}(P,\mathcal{X})$ is NP-hard, which means that $\mathrm{Dec}(P,\mathcal{X})$ is coNP-complete, since it belongs to coNP.

It is obvious that any incremental polynomial-time algorithm for generating $\mathcal{V}(P)$ is also output-polynomial. Perhaps, the next statement, stated implicitly in [39], is less obvious.

Proposition 1.1. *The following three claims are equivalent*:

(i) $\mathrm{Dec}(P,\mathcal{X})$ *is polynomial-time solvable.*
(ii) $\mathrm{Gen}(P)$ *is solvable in incremental polynomial.*
(iii) $\mathrm{Gen}(P)$ *is solvable in output polynomial-time.*

PROOF. Since (ii) $\Rightarrow$ (iii) clearly holds, we show that (i) $\Rightarrow$ (ii) and (iii) $\Rightarrow$ (i).

(i) $\Rightarrow$ (ii): We show that $\mathrm{Inc\,Gen}(P,\mathcal{X})$can be solved, i.e., a new vertex in $\mathcal{V}(P)\setminus\mathcal{X}$ can be found, by at most m calls to the decision problem $\mathrm{Dec}(P,\mathcal{X})$.

Let $\mathcal{J}$ be an algorithm that solves $\mathrm{Dec}(P,\mathcal{X})$. For $I\subseteq[m]\overset{\mathrm{def}}{=}\{1,\ldots,m\}$, denote respectively by A_I and b_I the submatrices of A and b formed by the rows $i\in I$, and let $\bar{I}=[m]\setminus I$. Define the polyhedron

$$P_I=\{x\in\mathbb{R}^n : A_Ix=b_I, A_{\bar{I}}\le b_{\bar{I}}\} \tag{1.3}$$

and $\mathcal{X}_I=\{x\in\mathcal{X}\mid x\in P_I\}$. We initialize $I=\emptyset$, and iterate the following, for $i=1,\ldots,m$:

Call $\mathcal{J}$ on $P_{I'}$ and $\mathcal{X}_{I'}$, where $I'=I\cup\{i\}$.

If $\mathcal{J}$ answers "No," then update $I:=I'$.

It is not difficult to see that a new $x\in\mathcal{V}(P)\setminus\mathcal{X}$ can be computed by solving linear equations $A_Ix=b_I$ for I obtained by the above procedure.

(iii) $\Rightarrow$ (i): Let $\mathcal{J}$ be an algorithm that solves $\mathrm{Gen}(P)$ in time $t(n,m,L,|\mathcal{V}(P)|)$. Given $\mathcal{X}\subseteq\mathcal{V}(P)$, we run $\mathcal{J}$, and stop as soon as one of the following conditions is satisfied:

(I) a vertex $x\in\mathcal{V}(P)\setminus\mathcal{X}$ is found,
(II) the running time of $\mathcal{J}$ exceeds $t(n,m,L,|\mathcal{X}|)$, or
(III) all vertices of P are found.

If (I) happens, we return "No." If (II) happens, we know that $|\mathcal{X}|<|\mathcal{V}(P)|$ and thus we return also "No." Otherwise, we can conclude that $\mathcal{X}=\mathcal{V}(P)$, and hence we return "Yes." The procedure above clearly implies (iii) $\Rightarrow$ (i). $\square$

In view of this proposition, if the decision problem is NP-hard, then no algorithm can generate all the elements of $\mathcal{V}(P)$ in incremental or total polynomial time, unless P = NP.

Given a polytope (i.e., bounded polyhedron) P by its representation (1.1), and a set $\mathcal{X}\subseteq\mathbb{R}^n$, checking if $P\subseteq\mathrm{conv}(\mathcal{X})$ was shown to be coNP-complete in [27]. If $\mathcal{X}\subseteq\mathcal{V}(P)$, then $\mathrm{conv}(\mathcal{X})\subseteq P$, and hence the problem becomes equivalent to problem $\mathrm{Dec}(P,\mathcal{X})$. More recently, it was shown [34] that, for general polyhedra, problem $\mathrm{Dec}(P,\mathcal{X})$ is NP-hard. Let us remark that polyhedra in the reduction of [34] are unbounded. Thus, it remains open whether the vertex generation problem for polytopes is also hard. On the other hand, it is well-known (as we shall also see below) that the problem of generating *jointly* the extreme points and extreme directions of a given polyhedron P (i.e., enumerating the elements of the set $\mathcal{V}(P)\cup$

$\mathcal{D}(P)$) is equivalent to generating the vertices $\mathcal{V}(P')$ of some polytope P' derived from P. In particular, if the vertex enumeration problem is polynomially solvable then the following statement will be somewhat surprising.

Proposition 1.2. *Let P be a polyhedron, given by its representation* (1.1). *Then*:

(i) *If vertex enumeration for polytopes is solvable in output polynomial time, then there is an incremental polynomial-time algorithm that outputs $\mathcal{V}(P)\cup\mathcal{D}(P)$ in the following order: all the elements of $\mathcal{D}(P)$ are generated first then all the elements of $\mathcal{V}(P)$.*

(ii) *Unless $P = NP$, there is no incremental polynomial-time algorithm that outputs $\mathcal{V}(P) \cup \mathcal{D}(P)$ such that all the elements of $\mathcal{V}(P)$ are generated before any element of $\mathcal{D}(P)$.*

Proof. The second statement follows from the NP-hardness of $\mathrm{Dec}(P, \mathcal{X})$ [34]. Let us prove the first one. Let $U = 2^{\mathrm{poly}(n,m,L)}$ be a strict upper bound on the ℓ_1-norm $\|x\|_1$ of any vertex x of P (such bounds are known to exists for rational polyhedra, see, e.g., [48]). Let P' be the polytope $P \cap \{x \in \mathbb{R}^n : -U \leq \mathbf{e}^T x \leq U\}$, where $\mathbf{e}$ is the vector of all ones. Then it can be verified that the set $\mathcal{V}(P) \cup \mathcal{D}(P)$ is in one-to-one correspondence with $\mathcal{V}(P')$, and furthermore that $\mathcal{V}(P) = \mathcal{V}(P') \cap \{x \in \mathbb{R}^n : \|x\|_1 < U\}$. It is also known that the extreme directions of P are in one-to-one correspondence with the vertices of the polytope $P'' = \{x \in \mathbb{R}^n : Ax \leq \mathbf{0}, c^T x = 1\}$, where $c \in \mathbb{R}^n$ is a vector not orthogonal to any extreme direction of P. Thus, in the first stage, we use our polynomial-time vertex enumeration routine to solve problem $\mathrm{Gen}(P'')$ and get all extreme directions of P, in time $\mathrm{poly}(n, m, L, |\mathcal{D}(P)|)$. In the second stage, we start with $\mathcal{X} = \mathcal{D}(P)$ and use the routine to solve $\mathrm{Inc\,Gen}(P', \mathcal{X})$.

The total time for the second stage is $\mathrm{poly}(n, m, L, |\mathcal{D}(P)| + |\mathcal{V}(P)|)$. $\square$

1.2. Monotone generation. We consider a *monotone* property[1] $\pi\colon 2^W \to \{0,1\}$ defined over the subsets of a finite set W: $\pi(X) \leq \pi(Y)$ whenever $X \subseteq Y \subseteq W$. We assume that there exists a polynomial-time evaluation oracle for π, that is, an algorithm that, given any $X \subseteq W$, determines in polynomial time in the size of W (and maybe some other input parameters), the value of $\pi(X) \in \{0, 1\}$. We denote by $\mathrm{Min}(\pi) \subseteq 2^W$ (respectively, $\mathrm{Max}(\pi) \subseteq 2^W$) the family of all *minimal* (respectively, *maximal*) subsets of W satisfying (respectively, not satisfying) a monotone property π. We will be interested in the generation of the families $\mathrm{Min}(\pi)$ (and/or $\mathrm{Max}(\pi)$), and so we can define, as before, the corresponding decision and generation problems $\mathrm{Dec}_{\mathrm{Min}(\pi)}(\pi, \mathcal{X})$, $\mathrm{Inc\,Gen}_{\mathrm{Min}(\pi)}(\pi, \mathcal{X})$, and $\mathrm{Gen}_{\mathrm{Min}(\pi)}(\pi)$.

Given a polyhedron P, defined as (1.1), one can express the vertex enumeration problem for P as a monotone generation problem, for instance, as follows. Let $W = [m]$ and for $I \subseteq W$, denote by P_I the polyhedron (1.3). Define the monotone property $\pi_1\colon 2^W \to \{0,1\}$ as follows

$$\pi_1(I) = 1 \iff P_I \neq \emptyset.$$

Then $\mathrm{Max}(\pi_1)$ is the family of *maximal tight feasible subsystems* of P, and it is not difficult to verify that they are in one-to-one correspondence with the vertices of P:

[1]We shall say that $X \subseteq W$ satisfies π if and only if $\pi(X) = 1$.

Fact 1. If $\mathcal{V}(P) \neq \emptyset$, then there exists a one-to-one correspondence between $\text{Max}(\pi_1)$ and $\mathcal{V}(P)$.

We now give another monotone formulation of the vertex enumeration problem. Consider a polyhedron $P = \{x \in \mathbb{R}^n : Ax = b, x \geq 0\}$ given in *standard form*, and let $\mathcal{A} \subseteq \mathbb{R}^m$ be a set of $n+1$ vectors in $\mathbb{R}^m$ consisting of the columns of A and $-b$. Define the monotone property $\pi_2 \colon 2^{\mathcal{A}} \to \{0,1\}$:

$$\pi_2(X) = 1 \iff \mathbf{0} \in \text{conv}(X). \tag{1.4}$$

Let us call the families $\text{Min}(\pi_2)$ and $\text{Max}(\pi_2)$ *simplices* and *anti-simplices*, respectively, with respect to $\mathcal{A}$. Then it turns out that the vertex enumeration problem for polytopes is equivalent to the generation of such simplices. More precisely, we have the following relationship.

Fact 2 ([34]). There exists a one-to-one correspondence between $\text{Min}(\pi_2)$ and $\mathcal{V}(P) \cup \mathcal{D}(P)$.

It is worth mentioning that if we change the monotone property above to be defined as: $\pi_3(X) = 1$ if and only if $\mathbf{0} \in \text{int}\big(\text{conv}(X)\big)$, where $\text{int}(Y)$ denotes the interior of a set Y, then we can say more about status of the problem. Namely, it was shown in [10] that $\text{Dec}_{\text{Max}(\pi_3)}(\pi_3, \mathcal{X})$ (and hence the generation of the so-called *anti-bodies*) is NP-hard. On the other hand, the generation of the family $\text{Min}(\pi)$ of minimal subsets of $\mathcal{A}$ that contain $\mathbf{0}$ in the interior of their convex hull (these are called *bodies* in [10]) turns out to be at least as hard as the hypergraph transversal generation problem, described in the next section.

1.3. Hypergraph transversals. Let $\mathcal{H} \subseteq 2^V$ be a hypergraph on a finite set of vertices V. A subset $X \subseteq V$ is said to be a transversal of $\mathcal{H}$ if $X \cap H \neq \emptyset$ for all $H \in \mathcal{H}$. We denote by $\mathcal{H}^d$ the transversal hypergraph of $\mathcal{H}$, i.e., the hypergraph consisting of all *minimal* transversals of $\mathcal{H}$. Clearly, if we define a monotone property $\pi \colon 2^V \to \{0,1\}$ by $\pi(X) = 1$ if and only if $X \cap H \neq \emptyset$ for all $H \in \mathcal{H}$, then we have a one-to-one correspondence between $\mathcal{H}^d$ and $\text{Min}(\pi)$. It turns out that the generation problem for many interesting monotone properties is reducible to problem $\text{Gen}_{\mathcal{H}^d}(\mathcal{H})$ of generating all minimal transversals of a hypergraph $\mathcal{H}$, see, e.g., [8, 23, 36].

The best known algorithm [24] for this problem runs in quasi-polynomial time $N^{o(\log N)}$, where $N = |V| + |\mathcal{H}| + |\mathcal{H}^d|$. No polynomial algorithm is known, though such algorithms exist for many special cases, e.g., for graphs or, more generally, for the hypergraphs of bounded edge-size [9, 21, 42, 43].

A similarity between the polytope-polyhedron and hypergraph transversal problems is pointed out in [40]. It is also mentioned there that the latter problem may be strictly between P and coNP, a conjecture attributed to Georg Gottlob.

2. Four generation techniques

In this section, we describe four methods that have been successfully applied to monotone generation problems. We then survey some of the known results about vertex enumeration and put them in the framework of these enumeration techniques. On the way, we also obtain some new results. Namely, we show in Section 3 that the method used for enumerating vertices of 0/1-polytopes is unlikely to extend for 0/1-polyhedra. We also give incremental polynomial-time algorithms for enumerating all vertices of polyhedra associated with 0/1-network matrices.

2.1. The supergraph (or graph search) approach. Let W be a finite set and assume that we are interested in generating the family $\mathcal{F}_\pi$ of all subsets of W satisfying a certain (not necessarily monotone) property $\pi\colon 2^W \to \{0,1\}$. This technique works by building and traversing a directed graph $\mathcal{G} = (\mathcal{F}_\pi, \mathcal{E})$, defined on the family $\mathcal{F}_\pi \subseteq 2^W$ to be generated. The arcs of $\mathcal{G}$ are defined by a neighborhood function $\mathcal{N}\colon \mathcal{F}_\pi \to 2^{\mathcal{F}_\pi}$ that to any $X \in \mathcal{F}_\pi$ assigns a set of its successors $\mathcal{N}(X)$ in $\mathcal{G}$. A special vertex $X_0 \in \mathcal{F}_\pi$ is identified from which all other vertices of $\mathcal{G}$ are reachable. The algorithm, shown in Algorithm 1, works by traversing, say, in the breadth-first search order, the vertices of $\mathcal{G}$, starting from X_0. If $\mathcal{G}$ is strongly connected then X_0 can be any vertex in $\mathcal{F}_\pi$.

The following facts are known about this approach (see, e.g., [32, 49]):

Proposition 2.1. (i) *If $\mathcal{N}(X)$ is polynomial-time computable for all $X \in \mathcal{F}_\pi$, then* GEN-A *yields a polynomial-delay algorithm for enumerating $\mathcal{F}_\pi$.*

(ii) *If $\mathcal{N}(X)$ can be generated in incremental polynomial time for all $X \in \mathcal{F}_\pi$, then* GEN-A *yields a incremental polynomial-time algorithm for enumerating $\mathcal{F}_\pi$.*

(iii) *If both $\mathcal{N}(X)$ and $\mathcal{N}^{-1}(X)$ are polynomial-time computable for all $X \in \mathcal{F}_\pi$, and $\mathcal{G}$ is a tree, then using depth-first search instead of breadth-first search in* GEN-A*, $\mathcal{F}_\pi$ can be generated with polynomial delay and polynomial space.*

We remark that if neighborhood function is polynomial-time computable, then we have $|\mathcal{N}(X)| \le \mathrm{poly}(|W|)$ for every $X \in \mathcal{F}_\pi$. For (iii), it is not difficult to see that $\mathcal{F}_\pi$ can be generated in incremental polynomial time if $\mathcal{N}(X)$ is polynomial-time computable and $\mathcal{G}$ is a tree. In order to obtain a polynomial-delay algorithm, $X \in \mathcal{F}_\pi$ is output just after the first visit to it, if the depth of X is odd; Otherwise, X is output just before coming back to the parent.

2.1.1. *The neighborhood operator for vertices of polyhedra.* Let P be a polyhedron given by (1.1). For a vertex $v \in \mathcal{V}(P)$, it is natural to define the neighbors

ALGORITHM 1. The supergraph method.

```
GEN-A($\mathcal{G}$)
Input: A supergraph $\mathcal{G}$ on the family $\mathcal{F}_\pi$ satisfying a property $\pi$
Output: The elements of family $\mathcal{F}_\pi$
 1: Find an initial vertex $X_0 \in \mathcal{F}_\pi$
 2: Initialize a queue $\mathcal{Q} = \{X_0\}$ and a dictionary of output vertices $\mathcal{D} = \{X_0\}$. {(*
    Perform a breadth-first search of $\mathcal{G}$ starting from $X_0$ *)}
 3: while $\mathcal{Q} \neq \emptyset$ do
 4:   Take the first vertex $X$ out of the queue $\mathcal{Q}$ and output $X$
 5:   for each $Y \in \mathcal{N}(X)$ do
 6:     if $Y \notin \mathcal{D}$ then
 7:       Insert it to $\mathcal{Q}$ and to $\mathcal{D}$
 8:     end if
 9:   end for
10: end while
```

of v as the "polyhedral" neighbors, that is

$$\mathcal{N}(v) = \{v' \in \mathcal{V}(P) : (v', v) \text{ forms an edge of } P\}.$$

As is well-known, the supergraph $\mathcal{G}$, defined with this neighborhood operator is strongly connected. Thus, by Proposition 2.1, enumerating the vertices of $\mathcal{V}(P)$ reduces to the neighborhood computation. Conversely, as observed in [46], the generation of $\mathcal{N}(v)$ for a given v is at least as hard as the vertex enumeration problem for polytopes. For, consider the generation of the vertices adjacent to the $(n+1)$st-unit vector $\mathbf{1}^{n+1}$ in the pyramid $\text{pyr}(x, P) \subseteq \mathbb{R}^{n+1}$ with base P and apex $\mathbf{1}^{n+1}$. However in some cases, as the two examples given below, the neighborhood computation can be done with polynomial delay or in incremental polynomial time.

2.1.2. *Simple polyhedra.* Recall that a polyhedron $P \subseteq \mathbb{R}^n$ is *simple* if each vertex of P is the intersection of exactly n facet-defining inequalities for P. For such polyhedra, the vertices are in one-to-one correspondence with the subsets of linearly independent tight inequalities (whose unique solution belongs to P), and hence the neighborhood operator can be computed in polynomial time. Indeed, let $v, v' \in \mathcal{V}(P)$ be two vertices of P, and $I, I' \subseteq [m]$ be the linearly independent tight inequalities of P, defining these vertices respectively. Then v and v' are neighbors if and only if $|I \setminus I'| = 1$. Thus for any vertex v, we have $|\mathcal{N}(v)| \leq n(m-n)$, and hence by Proposition 2.1(i), Traversal($\mathcal{G}$) enumerates $\mathcal{V}(P)$ with polynomial delay. It was furthermore observed by Avis and Fukuda [3] that the supergraph $\mathcal{G}$ can be turned into a tree as follows. Fix an arbitrary vector $c \in \mathbb{R}^n$, for which there is $v_0 \in \mathcal{V}(P)$ such that $cv_0 < cx$ for all $x \in P$. Then starting from any vertex $v \in \mathcal{V}(P)$ and using the simplex method with any anti-cyclic rule, there is a unique path from v to v_0 on P. This defines a tree with root v_0, for which the children of a given vertex v can be generated by finding first all candidates v' as above, but only keeping the vertices v' such that v is obtained from v' by a single simplex pivoting operation. By traversing the tree in depth-first search, we obtain thus by Proposition 2.1(iii) that the vertices of P can be enumerated with polynomial delay and space. Clearly, the same can also be said about polyhedra in which every vertex is the intersection of at most $n + \delta$ tight inequalities, for some constant δ, since a vertex can correspond to at most $\binom{n+\delta}{n} \leq O(n^\delta)$ sets of linearly independent tight inequalities.

2.1.3. *Flow polyhedra.* Let $G = (V, E)$ be a directed graph with vertex set V and arc set E, and let $A \in \{-1, 0, 1\}^{V \times E}$ be the the vertex-arc incidence matrix of G, that is

$$a_{u,e} = \begin{cases} 1 & \text{if arc } e \text{ enters } u, \\ -1 & \text{if arc } e \text{ leaves } u, \\ 0 & \text{otherwise} \end{cases}$$

for a vertex $u \in V$ and an arc $e \in E$. For $b \in \mathbb{R}^V$, let $P = P(A, b) = \{x \in \mathbb{R}^E : Ax = b, x \geq 0\}$ be the *flow polyhedron* associated with G. Provan [46] considered such a class of polyhedra (even with upper and lower bounds on the variables $l \leq x \leq c$, for some $l, c \in \mathbb{R}^E$) and showed that they can be highly degenerate. Furthermore, he gave an incremental polynomial time algorithm for enumerating the vertices using the supergraph approach as follows. Let $v \in \mathcal{V}(P)$ be a given vertex of P, and construct a graph $G' = (V', E')$ from G by contracting all arcs e with $v_e > 0$. Then the polyhedral edges adjacent to v are in one-to-one correspondence with the directed cycles of G', which can be enumerated with polynomial delay [47].

This gives an incremental polynomial time procedure for enumerating $\mathcal{V}(P)\cup\mathcal{D}(P)$ (cf. Proposition 2.1(ii)). Furthermore, one can also list the set of vertices $\mathcal{V}(P)$ by observing that an unbounded edge (that is, an extreme direction) of P corresponds to a directed cycle of G', for which all the contracted arcs, that have both vertices on the cycle, are on the same direction of the cycle. Let $E'' = \{e \in E : v_e > 0\}$, and for an arc $e \in E''$, let G_e be the graph obtained from G by contracting all arcs in $E'' - e$. Let $\mathcal{P}_{(u,w)}$ be the set of directed paths from u to w in $G_{(u,w)}$, which can be found with polynomial delay [47]. Then the set of bounded edges adjacent to v (that correspond to neighbors $v' \in \mathcal{N}(v)$) are in one-to-one correspondence with the set $\bigcup_{e\in E}\mathcal{P}_e$ (see [46] for more details). Since each element of $\mathcal{N}(v)$ can be found at most E times, we get an output polynomial algorithm for enumerating $\mathcal{N}(v)$, which can be turned into an incremental polynomial one, by Proposition 1.1.

A similar result was also obtained in [46] for the polyhedra $P(A^T, b)$ obtained from the transpose of A and $b \in \mathbb{R}^E$. This result was furthermore generalized in [1] to any matrix A with at most two nonzero entries in each row or with at most two nonzero entries in each column.

2.2. Using transversal generation.

2.2.1. *Polyhedra with* 0/1*-matrices and* 0/1*-vertices.* For $A \in \mathbb{R}^{m\times n}$ and $b \in \mathbb{R}^m$, let

$$P(A,b) = \{x \in \mathbb{R}^n \mid Ax \geq b, x \geq 0\}. \tag{2.1}$$

For a matrix $A \in \{0,1\}^{m\times n}$, let $\mathcal{H}(A) \subseteq 2^{[n]}$ be a hypergraph such that the characteristic vectors of hyperedges are the rows of A. We denote by $\mathbf{1}_m$ the m-dimensional vector all of whose components are ones. The following fact relates the vertices of an integral polyhedron $P(A, \mathbf{1}_m)$ to the minimal transversals of $\mathcal{H}(A)$.

Proposition 2.2 ([38])**.** *Let A be an $m\times n$ 0/1-matrix such that the polyhedron $P = P(A, \mathbf{1}_m)$ has only integral vertices. Then the vertices of P are in one-to-one correspondence with the minimal transversals of the hypergraph $\mathcal{H}(A)$.*

By the above proposition, the problem of enumerating the vertices of the polyhedron $P(A, \mathbf{1}_m)$, when A is a 0/1-matrix and $\mathcal{V}(P) \subseteq \{0,1\}^n$, reduces to finding all minimal transversals of the hypergraph $\mathcal{H}(A)$. As an immediate consequence of this and the result of [24], we obtain the following statement.

Corollary 2.3. *Let A be an $m \times n$ 0/1-matrix such that the polyhedron $P(A, \mathbf{1}_m)$ has only integral vertices. Then the vertices of $P(A, \mathbf{1}_m)$ can be enumerated in incremental quasi-polynomial time.*

For example, if the matrix A is *totally unimodular*, then the polyhedron $P(A, \mathbf{1}_m)$ has integral vertices which can be enumerated in incremental quasi-polynomial time. As a further interesting special case, a matrix $A \in \{0,1\}^{m\times n}$ is said to be a *network matrix*, if there exists a directed tree [2] $\mathbf{T}$ such that the rows of A one-to-one correspond to the arcs in $\mathbf{T}$ and each column of A is the characteristic vector of a directed path[3] in $\mathbf{T}$. Such a representation of a network matrix can be found from A in polynomial time (see, e.g., [48]).

[2]We say that a directed graph G is a *directed tree* if the underlying graph of G (i.e., the undirected graph obtained from G by ignoring orientation of arcs) is a tree.

[3]i.e., a set of arcs $\{(v_1, v_2), (v_2, v_3), \ldots (v_{k-1}, v_k)\}$, where $v_1, \ldots, v_k$ are vertices of $\mathbf{T}$.

It is well-known that a network matrix is totally unimodular, and hence Corollary 2.3 implies that the vertices of the polyhedron $P(A, \mathbf{1}_m)$ can be enumerated in incremental quasi-polynomial time. In Section 3, we show the following stronger results for polyhedra associated with 0/1-network matrices.

Theorem 2.4. *Let $A \in \{0,1\}^{m\times n}$ be a network matrix. Then we have*:

(i) *The vertices of $P(A, \mathbf{1}_m)$ can be enumerated in incremental polynomial time using polynomial space.*
(ii) *The vertices of $P(A^T, \mathbf{1}_n)$ can be enumerated in incremental polynomial time using polynomial space.*

In the next section, we relate the problems of enumerating vertices of the polyhedra $P(A, \mathbf{1}_m)$ and $P(A^T, \mathbf{1}_n)$ for a 0/1-network matrix A to two other enumeration problems on directed trees. This will turn out to be also useful in the NP-hardness proof of Section 2.3.4.

2.2.2. *Two enumeration problems on directed trees.* Given a directed tree $\mathbf{T} = (V, E)$ and a set of n directed paths on $\mathbf{T}$, defined by source-sink pairs $\mathcal{P} = \{(s_i, t_i) \mid s_i, t_i \in V \text{ for } i = 1, \dots, n\}$, let us call a *minimal path cover* any minimal collection of paths $X \subseteq \mathcal{P}$ whose union covers all the arcs of $\mathbf{T}$. Let us further call a *minimal cut conjunction* any minimal collection of arcs $X \subseteq E$ whose removal leaves no path between s_i and t_i for $i = 1, \dots, n$.

The following statement is clear form the definitions, Proposition 2.2, and the fact that a network matrix is totally unimodular.

Proposition 2.5. *Let $A \in \{0,1\}^{m\times n}$ be a network matrix, defined by a tree $\mathbf{T}$ and a collection of directed paths $\mathcal{P}$. Then*:

(i) *The vertices of $P(A, \mathbf{1}_m)$ are in one-to-one correspondence with the minimal path covers for $(\mathbf{T}, \mathcal{P})$.*
(ii) *The vertices of $P(A^T, \mathbf{1}_n)$ are in one-to-one correspondence with the minimal cut conjunctions for $(\mathbf{T}, \mathcal{P})$.*

In view of Proposition 2.5, the two parts of Theorem 2.4 follow respectively from the following two lemmas, proved in Section 3.

Lemma 2.6. *Given a directed tree $\mathbf{T}$, and a collection of directed paths $\mathcal{P}$, the family of minimal path covers with respect to $(\mathbf{T}, \mathcal{P})$ can be enumerated in incremental polynomial time using polynomial space.*

Lemma 2.7. *Given a directed tree $\mathbf{T}$, and a collection of directed paths $\mathcal{P}$, the family of minimal cut conjunctions with respect to $(\mathbf{T}, \mathcal{P})$ can be enumerated in incremental polynomial time using polynomial space.*

We remark that an incremental polynomial time algorithm exists [35] for the more general problem of enumerating cut conjunctions in undirected graphs: Given an undirected graph $G = (V, E)$, and a collection $B = \{(s_1, t_1), \dots, (s_k, t_k)\}$ of k vertex pairs $s_i, t_i \in V$, enumerate all minimal edge sets $X \subseteq E$ such that for all $i = 1, \dots, k$, vertices s_i and t_i are disconnected in $G' = (V, E \setminus X)$. This is obtained using the supergraph approach. However, in contrast to the one presented in Section 3.2, the space used by the algorithm is not polynomial.

2.2.3. *Transversal-bounded polyhedra.* Recall that the vertices of a polytope $P = \{x \in \mathbb{R}^n : Ax = b, x \geq 0\}$ can be regarded as the minimal sets satisfying a monotone property π_2, given by (1.4). Let us call a polyhedron *transversal-bounded* (or *dual-bounded*) [8, 16] if there exists a (quasi-)polynomial $q\colon \mathbb{R}_+ \to \mathbb{R}_+$, for which the following inequality holds

$$|\operatorname{Max}(\pi_2)| \le q(n, m, L, |\operatorname{Min}(\pi_2)|). \tag{2.2}$$

It follows from the results of [5, 30] that, for transversal-bounded polyhedra, problem $\operatorname{Gen}(P)$ is (quasi-)polynomially reducible to the hypergraph transversal generation problem, and thus is solvable in incremental quasi-polynomial time. Probably, the only example known so far to satisfy (2.2) is the class of simple polytopes [4] for which it is known that (see [19] and also [29, 45]) $|\operatorname{Max}(\pi_1)| \le (n+1)|\operatorname{Min}(\pi_1)|$. Although an inequality of this form does not hold in general (see [10]), it will be interesting to see if there are other types of polyhedra that belong to this class.

2.3. Flashlight approach and its applications.

2.3.1. *Flashlight* (*or backtracking*) *method.* Let $W = \{1, 2, \dots, |W|\}$, and suppose that we want to enumerate all elements of a family $\mathcal{F}_\pi$ of subsets of W satisfying a given (not necessarily monotone) property π, where $\mathcal{F}_\pi \ne \emptyset$. This method works by building a binary search tree of depth $|W|$ whose leaves ans some of its internal nodes contain the elements of the family $\mathcal{F}_\pi$. Each node of the tree is identified with an ordered pair (S_1, S_2) of two disjoint subsets $S_1, S_2 \subseteq W$, and at the root of the tree, we have $S_1 = S_2 = \emptyset$. The two children of an internal node (S_1, S_2) of the tree are defined as follows. We choose an element $e \in W \setminus (S_1 \cup S_2)$ such that there is an $X \in \mathcal{F}_\pi$, satisfying $X \supseteq S_1 \cup \{e\}$ and $X \cap S_2 = \emptyset$. If no such element can be found, then the current node is a leaf. Otherwise, the left child of the node (S_1, S_2) is identified with $(S_1 \cup \{e\}, S_2)$. Analogously, the right child of the node (S_1, S_2) is $(S_1, S_2 \cup \{e\})$, provided that there is an $X \in \mathcal{F}_\pi$, such that $X \supseteq S_1$ and $X \cap (S_2 \cup \{e\}) = \emptyset$. A formal description of the method is given in Algorithm 2 (see [47] for general background on backtracking algorithms). We assume $\emptyset \notin \mathcal{F}_\pi$.

Clearly, for this method to work in polynomial time, we need to be able to perform the following check in polynomial time:

$\operatorname{Ext}(\pi, S_1, S_2)$**:** Given two disjoint subsets $S_1, S_2 \subseteq W$, does there exist a set $X \in \mathcal{F}_\pi$, such that $X \supseteq S_1$ and $X \cap S_2 = \emptyset$?

The performance of this method is summarized in the following statement.

Proposition 2.8. *If* $\operatorname{Ext}(\pi, S_1, S_2)$ *is solvable in polynomial time for any given disjoint sets* $S_1, S_2 \subseteq W$*, then* GEN-B *enumerates the family* $\mathcal{F}_\pi$ *with polynomial delay using polynomial space* (*even in lexicographic order, if some ordering on* W *is given*)*.*

In general, the check $\operatorname{Ext}(\pi, S_1, S_2)$ is NP-hard, but in some cases, as the ones described below, it can be performed in polynomial time. Sometimes, it is also possible to perform the check in polynomial time, provided that we do the extension from the set S_1 to $S_1 \cup \{e\}$ in a more careful way. More precisely, let $\mathcal{F}'_\pi \subseteq 2^E$ be a family of sets, such that:

(F1) $\mathcal{F}'_\pi \supseteq \mathcal{F}_\pi$,
(F2) for every non-empty $X \in \mathcal{F}'_\pi$, there exists an element $e \in X$ such that $X \setminus \{e\} \in \mathcal{F}'_\pi$ (in particular, $\emptyset \in \mathcal{F}'_\pi$), and
(F3) we can test in polynomial time if a given set $X \in \mathcal{F}'_\pi$ is contained in $\mathcal{F}_\pi$.

[4]each vertex of which is *non-degenerate* and has exactly m positive components, if the row rank of A is m.

Algorithm 2. The flashlight method.

Procedure GEN − B(π, S_1, S_2):
Input: A property $\pi\colon 2^W \to \{0,1\}$ and two disjoint sets $S_1, S_2 \subseteq W$
Output: The family $\mathcal{F}_\pi$ of all subsets X of W satisfying $X \supseteq S_1$, $X \cap S_2 = \emptyset$, and $\pi(X) = 1$

```
1: if there is an $e \in W \setminus (S_1 \cup S_2)$ s.t. Ext$(\pi, S_1 \cup \{e\}, S_2)$ is feasible then
2:   if If $S_1 \cup \{e\} \in \mathcal{F}_\pi$ then
3:     output $S_1 \cup \{e\}$
4:     GEN − B$(\pi, S_1 \cup \{e\}, S_2)$
5:     GEN − B$(\pi, S_1, S_2 \cup \{e\})$
6:   end if
7: end if
8: return
```

In the backtracking procedure, if we always maintain the invariant $S_1 \in \mathcal{F}'_\pi$, Then checking Ext(π, S_1, S_2) could be easier, see the example in Section 2.3.3.

2.3.2. 0/1-*polytopes.* Bussieck and Lübbecke [18] used the flashlight method to show the strong P-enumerability of the vertex set of 0/1-polytopes (more generally, of polytopes that are combinatorially equivalent with 0/1-polytopes). Recall that a polyhedron P is 0/1 if $\mathcal{V}(P) \subseteq \{0,1\}^n$. Let $W = [n]$ and $\pi\colon 2^W \to \{0,1\}$ be defined as follows: for $X \subseteq W$, $\pi(X) = 1$ if and only if $X \subseteq [n]$ is the support set of a vertex. Then Problem Ext(π, S_1, S_2) calls for the following check:

Ext(π, S_1, S_2)**:** Given a 0/1-polyhedron P defined by (2.1) and two disjoint sets of variables $S_1, S_2 \subseteq [n]$, determine if there is a vertex x of P such that $x_i = 1$ for all $i \in S_1$ and $x_i = 0$ for all $i \in S_2$.

If the polyhedron P is bounded, i.e., P is a 0/1-polytope, then the extension problem above is equivalent to checking if the polytope

$$P' = \{x \in P \mid x_i = 1 \text{ for all } i \in S_1, \text{ and } x_i = 0 \text{ for all } i \in S_2\}.$$

is non-empty, and hence it can be checked in polynomial time by solving a linear programming problem. Thus Proposition 1.1 implies that $\mathcal{V}(P)$ is strongly P-enumerable in this case. Note that this remains true even if the polytope is given by a polynomial-time separation oracle. However, the problem seems to be intractable for unbounded 0/1-polyhedra (see Theorem 2.11).

2.3.3. *The perfect* 2*-matchings polytope.* Let $G = (V, E)$ be a graph. Consider the polytope

$$P(G) = \{x \in \mathbb{R}^E \mid Ax = \mathbf{1}_n, x \geq 0\},$$

where $A \in \{0,1\}^{V \times E}$ denotes the vertex-edge incidence matrix of G, and $n = |V|$. When G is bipartite, the vertices of $P(G)$ are in one-to-one correspondence with the perfect matchings of G, and the result of the the previous section implies that $\mathcal{V}(P)$ can be enumerated with polynomial delay. More efficient algorithms are known [25, 26, 52, 53].

For non-bipartite graphs G, it is well-know that the vertices of $P(G)$ are half-integral [41] (i.e., the components of each vertex are in $\{0, 1, 1/2\}$), and that they correspond to the *basic perfect* 2*-matchings* of G, i.e., subsets of edges that form a cover of the vertices with vertex-disjoint edges and vertex-disjoint odd cycles. A (not necessarily basic) perfect 2-matchings of G is a subset of edges that covers the vertices of G with vertex-disjoint edges and (even or odd) cycles. Denote respectively by $\mathcal{M}_2(G)$ and $\mathcal{M}_2'(G)$ the families of perfect 2-matchings and basic perfect 2-matchings of a graph G. It was shown in [6] that the family $\mathcal{M}_2(G)$ can be enumerated with polynomial delay, and the family $\mathcal{M}_2'(G)$ can be enumerated in incremental polynomial time. To illustrate the application of the flashlight method, we include the proof of Lemma 2.9 from [6].

Lemma 2.9 ([6])**.** *All perfect* 2*-matchings of a graph* G *can be generated with polynomial delay.*

Proof. We use the flashlight method with a slight modification. For $X \subseteq E$, let $\pi(X)$ be the property that the graph (V, X) has a perfect 2-matchings. Then $\mathcal{F}_\pi = \mathcal{M}_2(G)$. Define

$$\mathcal{F}_\pi' = \{X \subseteq E \mid \text{ the graph } (V, X) \text{ is a vertex-disjoint union} \\ \text{of some cycles, some edges, and possibly a single path}\}.$$

It is easy to check if conditions (F1), (F2) and (F3) are satisfied. Given $S_1 \in \mathcal{F}_\pi'$, $S_2 \subseteq E$, we modify the basic approach described in Section 2.3 in two ways. First, when we consider a new edge $e \in E \setminus (S_1 \cup S_2)$ to be added to S_1, we first try an edge incident to an endpoint of the path in S_1, if this path exists. If there is no such path in S_1, then any edge $e \in E \setminus (S_1 \cup S_2)$ can be chosen and defined to be a path of length one in $S_1 \cup \{e\}$. Second, when we backtrack on an edge e defining a path of length one in S_1, we redefine S_1 by considering e as a single edge rather than a path of length one. Now it remains to verify that $\mathrm{Ext}(\pi, S_1, S_2)$ can be checked in polynomial time. Given $S_1 \in \mathcal{F}_\pi'$, $S_2 \subseteq E$, and an edge $e \in E \setminus (S_1 \cup S_2)$, chosen as above, such that $S_1 \cup \{e\} \in \mathcal{F}_\pi'$, we can check in polynomial time whether there is an $X \in \mathcal{M}_2(G)$ such that $X \supseteq S_1 \cup \{e\}$ and $X \cap S_2 = \emptyset$ in the following way. First, we delete from G all edges in S_2, and all vertices incident to edges in S_1, except the end-points x and y of the single path $\mathbf{P}$ in S_1. Let us call the resulting graph G'. Then, we construct an auxiliary bipartite graph G^b from G' as follows (see [41]). For every vertex $v \neq x, y$ of G' we define two vertices v' and v'' in G^b. In addition, B^b also contains two other vertices x' and y''. For every edge $\{u, v\}$ in G', with $\{u, v\} \cap \{x, y\} = 0$, we define two edges $\{u', v''\}$ and $\{u'', v'\}$ in G^b. For each edge $\{x, u\}$ in G', we introduce an edge $\{u', x''\}$ in G^b, and for each edge $\{u, y\}$ in G', we introduce an edge $\{u', y''\}$ in G^b. It is easy to see that there is an $X \in \mathcal{M}_2(G)$ such that $X \supseteq S_1 \cup \{e\}$ and $X \cap S_2 = \emptyset$ if and only if there is a perfect matching in G^b. □

Lemma 2.10 ([6])**.** *For a graph* $G = (V, E)$*, we have*

$$|\mathcal{M}_2(G)| \leq \binom{|\mathcal{M}_2'(G)| + 1}{2}. \tag{2.3}$$

Thus by generating all perfect 2-matchings of G and discarding the non-basic ones, we can generate all basic perfect 2-matchings. By Lemma 2.10, the total time for this generation is polynomial in $|V|$, $|E|$, and $|\mathcal{M}_2'(G)|$. By Proposition 1.1, we get an incremental polynomial-time algorithm for enumerating $\mathcal{V}(P(G))$.

2.3.4. *NP-hardness of flashlight for enumerating vertices of* 0/1*-polyhedra.* It is natural to ask whether the same method used for generating the vertices of 0/1-polytopes can be extended to polyhedra with 0/1-vertices. In this section we answer this question in the negative: we show that the extension problem, on which the efficiency of this method relies, is generally NP-hard. Our reduction will use polyhedra associated with network matrices of the form (2.1). Note that, if the vertices of a polyhedron $P = P(A, \mathbf{1}_m)$, defined by (2.1), are integral, then the vertices of the polytope $P \cap \{0,1\}^n$ correspond to the transversals of the hypergraph $\mathcal{H}(A)$, which might be exponentially larger in cardinality than the minimal transversals. More directly, we have the following negative results.

Theorem 2.11. *Let A is an $m \times n$ 0/1-network matrix. Then we have*:

(i) *For a set $S \subseteq [n]$, problem* $\mathrm{Ext}(P(A, \mathbf{1}_m), S, \emptyset)$ *is NP-complete.*
(ii) *For a set $S \subseteq [m]$, problem* $\mathrm{Ext}(P(A^T, \mathbf{1}_n), S, \emptyset)$ *is NP-complete.*

Proof. We reduce the following monotone satisfiability problem, which is known to be NP-complete [28] (see also Section A.2.1), to the two problems.

Problem Monotone SAT

Input: A conjunctive normal form (CNF) $\phi(x_1, \ldots, x_N) = C_1 \wedge \cdots \wedge C_M$, where each C_j, $j = 1, \ldots, M'$, is a disjunction of some literals in $\{x_1, \ldots, x_N\}$, and each C_j, $j = M'+1, \ldots, M$, is a disjunction of some literals in $\{\overline{x}_1, \ldots, \overline{x}_N\}$.

Question: Is there a satisfying truth assignment for CNF ϕ, i.e., $x \in \{0,1\}^N$ such that $\phi(x) = 1$?

(i) $\mathrm{Ext}(P(A, \mathbf{1}_m), S, \emptyset)$: Given a CNF ϕ, we define a network matrix A by constructing a directed tree $\mathbf{T} = (V, E)$ and a set $\mathcal{P}$ of directed paths in $\mathbf{T}$ as follows:

$$\begin{aligned}
V &= \{u_i', u_i'' \mid i \in [N]\} \cup \{c_j \mid j \in [M]\} \cup \{u_0\} \\
E &= \{(u_i', u_0), (u_0, u_i'') \mid i \in [N]\} \cup \{(u_0, c_j) \mid j \in [M']\} \\
&\quad \cup \big\{(c_j, u_0) \mid j \in \{M'+1, \ldots, M\}\big\} \qquad (2.4) \\
\mathcal{P} &= \{(u_i', u_0, u_i'') \mid i \in [N]\} \cup \{(u_i', u_0, c_j) \mid x_i \in C_j\} \\
&\quad \cup \{(c_j, u_0, u_i'') \mid \overline{x}_i \in C_j\}
\end{aligned}$$

Here vertices u_i' and u_i'', $i \in [N]$, correspond to positive and negative literals x_i and $\overline{x}_i$, respectively, and c_j, $j \in [M]$ corresponds to clause C_j in ϕ. Finally, we define the subfamily S of $\mathcal{P}$ by

$$S = \{(u_i', u_0, u_i'') \mid i \in [N]\}$$

and claim that $\mathrm{Ext}(P(A, \mathbf{1}_n), S, \emptyset)$ is equivalent to Monotone SAT.

From Proposition 1.2, we note that $\mathrm{Ext}(P(A, \mathbf{1}_n), S, \emptyset)$ is to check if S can be extended to a minimal path cover for $(\mathbf{T}, \mathcal{P})$, i.e., a minimal family of paths whose union contains all the arcs in E. Thus, to see our claim, we show that ϕ is satisfiable if and only if S is extendable to a minimal path cover for $(\mathbf{T}, \mathcal{P})$.

Let X be such a minimal extension, and let us define an assignment by setting $x_i := 1$ if and only if (u_i', u_0) is covered with $X - S$. Since the minimality of X implies that any path in S is not covered with $X - S$, (u_0, u_i'') is covered with $X - S$ only if $x_i = 0$. Now, since X is a path cover, for each $j = 1, \ldots, M'$, (u_0, c_j) is covered with some path in X, and hence C_j contains a literal x_i with value 1. Similarly, for each $j = M'+1, \ldots, M$, (c_j, u_0) is covered with some path in X, and hence C_j contains a literal $\overline{x}_i$ with value 0. These imply that CNF ϕ is satisfiable.

Conversely, from any satisfying assignment x for ϕ, we can construct a minimal path cover X that contains S by

$$X = S \cup \{(u_i', u_0, c_j) \mid i \text{ is the least index s.t. } x_i = 1 \text{ and } x_i \in C_j \text{ for } j \in [M']\} \\ \cup \big\{(c_j, u_0, u_i'') \mid i \text{ is the least index s.t. } x_i = 0 \text{ and } \overline{x}_i \in C_j \\ \text{for } j \in \{M'+1, \dots, M\}\big\}.$$

This completes the proof of (i).

(ii) $\mathrm{Ext}(P(A^T, \mathbf{1}_n), S, \emptyset)$: Given a CNF ϕ, we define a network matrix A (i.e., a directed tree $\mathbf{T} = (V, E)$ and a set $\mathcal{P}$ of directed paths in $\mathbf{T}$) by (2.4). Moreover, we define the subset S of arcs by

$$S = \{(u_0, c_j) \mid j \in [M']\} \cup \big\{(c_j, u_0) \mid j \in \{M'+1, \dots, M\}\big\},$$

and claim that $\mathrm{Ext}(P(A^T, \mathbf{1}_m), S, \emptyset)$ is equivalent to MONOTONE SAT.

From Proposition 1.2, we note that $\mathrm{Ext}(P(A^T, \mathbf{1}_m), S, \emptyset)$ is to check if S can be extended to a minimal cut conjunction for5 $(\mathbf{T}, \mathcal{P})$, i.e., a minimal set of arcs that hits every directed path in $\mathcal{P}$. Thus, to see our claim, we show that ϕ is satisfiable if and only if S is extendable to a minimal cut conjunction for $(\mathbf{T}, \mathcal{P})$.

Let X be such a minimal extension. Then clearly, it can be obtained from S and exactly one of the two arcs (u_i', u_0) and (u_0, u_i'') for every $i \in [N]$ (no other arc is contained in X). This defines an assignment by setting $x_i := 1$ if and only if $(u_i', u_0) \notin X$ (i.e., $(u_0, u_i'') \in X$). Now the minimality of X implies that for each $j = 1, \dots, M'$, there is a path (u_i', u_0, c_j) in $\mathcal{P}$ such that $(u_i', u_0) \notin X$, and and similarly, for $j = M'+1, \dots, M$, there is a path (c_j, u_0, u_i'') in $\mathcal{P}$ such that $(u_0, u_i'') \notin X$. This implies that CNF ϕ is satisfiable.

Conversely, from any satisfying assignment x for ϕ, we can construct a minimal cut conjunction X that contains S by

$$X = S \cup \{(u_0, u_i'') \mid x_i = 1\} \cup \{(u_i', u_0) \mid x_i = 0\}. \qquad \square$$

We conclude with the following remark. If flashlight works it results in a polynomial delay algorithm, while we get only an incremental polynomial one using the supergraph approach. However, for this reason, the flashlight subroutine is frequently NP-hard.

For example, for the transversal hypergraph problem, the flashlight technique calls the following decision subproblem. Given a hypergraph H and a subset of its vertices X, check whether X can be extended to a minimal transversal of H. A simple criterion is given in [14], see also [7]. However, the corresponding conditions are NP-hard to verify [14] already for graphs [7] unless the size of X is bounded by a constant.

The same happens for many generation problems on graphs. Given a (directed) graph $G = (V, E)$, a pair of vertices $s, t \in V$, and a subset $X \subseteq E$, it is NP-hard to decide whether X can be extended to a simple (directed) cycle, or to an s, t (directed) path, or to a minimal s, t-cut in G [37].

2.4. The projection technique. The proof of Lemma 2.6 and 2.7 is based on the following enumeration technique, developed originally in [51] (see also [32] and [39]). Let $W = [w] = \{1, \dots, w\}$ be a finite set of elements, and let π be a monotone property defined over 2^W. Here we assume that W satisfies π. Let $\mathcal{F}_\pi$ be the family of minimal subsets satisfying π. By assumption, we have $\mathcal{F}_\pi \neq \emptyset$.

Algorithm 3. The projection method.

```
Procedure GEN − C(π, i, X):
Input: A monotone property $\pi$, an index $i \in [w]$, and an $i$-minimal satisfying set
    $X \in \mathcal{F}_\pi^i$.
Output: The family $\mathcal{F}_\pi$ of all minimal subsets of $[w]$ satisfying $\pi$.
 1: if $i = w + 1$ then
 2:   output $X$;
 3: else
 4:   if $X \setminus \{i\}$ is satisfies $\pi$ then
 5:     $\mathrm{GEN-C}(\pi, i+1, X \setminus \{i\})$;
 6:   else
 7:     $\mathrm{GEN-C}(\pi, i+1, X)$;
 8:     for each minimal set $Y \in \mathcal{F}_\pi(i, X)$ do
 9:       if $X \cup Y \setminus \{i\} \in \mathcal{F}_\pi^{i+1}$ then
10:         Compute the lexicographically largest set $Z \subseteq X \cup Y$ s.t. $Z \in \mathcal{F}_\pi^i$.
11:         if $Z = X$ then
12:           $\mathrm{GEN-C}(\pi, i+1, X \cup Y \setminus \{i\})$;
13:         end if
14:       end if
15:     end for
16:   end if
17: end if
```

For $i = 1, \dots, w$, denote by $[i : w]$ the set $\{i, i+1, \dots, w\}$, where we define $[w+1 : w] = \emptyset$. By definition, we have $[1 : i] = [i]$. We shall say that a set $X \subseteq W$ *i-minimally satisfies* π if $X \supseteq [i : w]$, X satisfies π, and $X \setminus \{j\}$ does not satisfy π for all $j \in X \cap [i-1]$. Thus, $(w+1)$-minimally satisfying sets are just the minimal satisfying sets, i.e., the ones in the family $\mathcal{F}_\pi$. For $i = 1, \dots, w$, denote by $\mathcal{F}_\pi^i$ the family of sets that i-minimally satisfy property π. Given $i \in W$ and $X \in \mathcal{F}_\pi^i$, denote by $\mathcal{F}_\pi(i, X)$ the family of minimal subsets $Y \subseteq [i-1]$, such that $X \cup Y \setminus \{i\}$ satisfies π.

Proposition 2.12 (See [22, 39]**).** *Let $\mathcal{F}_\pi$, $\mathcal{F}_\pi^i$, and $\mathcal{F}_\pi(i, X)$ be defined as above. Then*:

(i) $|\mathcal{F}_\pi^i| \le |\mathcal{F}_\pi|$ *holds for all* $i \in [w+1]$.
(ii) $|\mathcal{F}_\pi(i, X)| \le |\mathcal{F}_\pi^{i+1}|$ *holds for all* $i \in [w]$ *and all* $X \in \mathcal{F}_\pi^i$.

Let us now formally describe a general procedure for generating all minimal sets that satisfy a monotone property π.

Given $i \in [w]$, and $X \in \mathcal{F}_\pi^i$, we assume in the algorithm below that the minimal sets in $\mathcal{F}_\pi(i, X)$ are computed by calling a process $\mathcal{A}(i, X)$, in which, once $\mathcal{A}(i, X)$ finds an element $Y \in \mathcal{F}_\pi(i, X)$, it returns control to the calling process $\mathrm{GEN}(\pi, i, X)$, and when called the next time, it returns the next element of $\mathcal{F}_\pi(i, X)$ that has not been generated yet, if such an element exits.

Proposition 2.13 ([9, 51])**.** *If the family $\mathcal{F}_\pi(i, X)$ can be enumerated in incremental polynomial time using polynomial space, for every $i \in [w]$ and every $X \in \mathcal{F}_\pi^i$, then so can the family $\mathcal{F}_\pi$ using* GEN − C.

3. Enumerating vertices of polyhedra associated with 0/1-network matrices

In this section, we show how to enumerate the families $\mathcal{F}_\pi(i, X)$, in the two cases of path covers and cut conjunctions on trees.

3.1. Enumerating minimal path covers. Let $\mathbf{T} = (V, E)$ be a directed tree, and let $\mathcal{P}$ be a collection of directed paths in $\mathbf{T}$. To apply the generation algorithm described in the previous section, we order the paths in $\mathcal{P}$ arbitrarily, say $\mathcal{P} = \{P_1, \ldots, P_n\}$, let $W = [n]$, and for each $X \subseteq W$, let π be the property that $\{P_i \mid i \in X\}$ is a path cover. For simplicity, we may sometimes refer to a path P_i directly by its index i. In this setting, we show the following.

Lemma 3.1. *Given an $i \in W$ and a set $X \in \mathcal{F}_\pi^i$ such that $X \setminus \{i\}$ is not a path cover, all elements of the family $\mathcal{F}_\pi(i, X)$ can be enumerated with delay $O(|V|\,|\mathcal{P}|)$ and space $O(|V| + |\mathcal{P}|)$.*

To generate the family $\mathcal{F}_\pi(i, X)$, let U be a set of arcs in E that are not covered with the paths in $X \setminus \{i\}$, and we define a hypergraph $\mathcal{H} \subseteq 2^U$ by $\mathcal{H} = \{P_j \cap U \mid j \in W \setminus X\}$. By definition, we have $U \subseteq P_i$ and $\mathcal{H}$ is interval, i.e., every hyperedge in $\mathcal{H}$ defines an interval (i.e., a subpath) in U, where U is regarded as a path obtained from P_i by contracting arcs which are contained in some path $P_j \in X \setminus \{i\}$.

Now, we can see that $\mathcal{F}_\pi(i, X)$ corresponds the family of all minimal covers of an interval hypergraph $\mathcal{H}$, and show that they can be enumerated efficiently, from which Lemma 3.1 follows.

The latter problem is known to be solvable in incremental polynomial time. Indeed, an interval hypergraph $\mathcal{H}$ is 2-*Helly*: a subset of hyperedges from $\mathcal{H}$ has a common vertex whenever every 2 hyperedges of this subset have one. The transposed hypergraph $\mathcal{H}^T$ is 2-conformal [4], and for this class for hypergraphs, it is known that the set of minimal transversals can be enumerated in incremental polynomial time [9]. Equivalently, if $\mathcal{H}^T$ is 2-conformal, all minimal covers for $\mathcal{H}$ can be enumerated in incremental polynomial time. Here we obtain the following stronger result, from which Lemma 3.1 follows.

Theorem 3.2. *Let $\mathcal{H} \subseteq 2^U$ be an interval hypergraph on a finite set U. Then all minimal covers of $\mathcal{H}$ can be generated with delay $O(|U|\,|\mathcal{H}|)$ and space $O(|U| + |\mathcal{H}|)$.*

Proof of Theorem 3.2. Let $U = \{1, \ldots, |U|\}$, and each hyperedge H in an interval hypergraph $\mathcal{H}$ is given by $I_H = [L_H : R_H]$, where $L_H \leq R_H$. Let $X = \{[L_1 : R_1], \ldots, [L_k : R_k]\}$ be a sub-hypergraph of $\mathcal{H}$, i.e., $X \subseteq \mathcal{H}$. If X is a minimal cover, then $L_i \neq L_j$ clearly holds for all i and j with $i \neq j$, since no interval in X contains another. Moreover, we have the following characterization.

Proposition 3.3. *Let $X = \{[L_1 : R_1], \ldots, [L_k : R_k]\}$ be a sub-hypergraph of $\mathcal{H} \subseteq 2^U$, such that $L_1 < L_2 < \cdots < L_k$. Then X is a minimal cover if and only if*

(i) $L_{i+1} \leq R_i + 1 < L_{i+2}$, *for $i = 1, \ldots, k-2$,*
(ii) $L_k \leq R_{k-1} + 1 < R_k + 1$, *and*
(iii) $[L_1 : R_k] = U$.

Proof. Suppose that $X \subseteq \mathcal{H}$ be a minimal cover. Then the minimality of X implies $R_{k-1} < R_k$. Note also that, for all $i = 1, \ldots, k-1$, we have $L_{i+1} \leq R_i + 1$, for otherwise the point $R_i + 1$ cannot be covered by any interval in X. Furthermore, for $i = 1, \ldots, k-2$, we have $L_{i+2} > R_i + 1$, for otherwise, $[L_i : R_i] \cup [L_{i+2} : R_{i+2}] \supseteq [L_{i+1} : R_{i+1}]$, contradicting the minimality of X. Since X is a cover of U, (i) and (ii) imply $[L_1 : R_k] = U$.

Conversely, if $X = \{[L_1 : R_1], \ldots, [L_k : R_k]\}$ satisfies properties (i)–(iii) stated in the proposition, then it is not difficult to see that X is a minimal cover of U. □

Let $\mathcal{F}$ be the family of minimal collections of intervals in $\mathcal{H}$ that cover U, and let $\mathcal{F}'$ be the family of all collections $X = \{[L_1 : R_1], \ldots, [L_k : R_k]\}$ of intervals in $\mathcal{H}$ satisfying properties (i)–(ii) of Proposition 3.3 such that $L_1 = 1$. By definition, we have $\mathcal{F} \subseteq \mathcal{F}'$, and any $X \in \mathcal{F}'$ is a minimal cover of $[1 : R_k] \subseteq U (= \{1, \ldots, |U|\})$. In our backtracking procedure, we shall always choose S_1 from $\mathcal{F}'$.

Given two disjoint subsets $S_1 = \{[L_1 : R_1], \ldots, [L_k : R_k]\} \in \mathcal{F}'$ and $S_2 \subseteq \mathcal{H}$, any interval $I = [L_{k+1} : R_{k+1}] \in \mathcal{H} \setminus (S_1 \cup S_2)$ satisfies $S_1 \cup \{I\} \in \mathcal{F}'$ if and only if $R_{k-1} + 1 < L_{k+1} \leq R_k + 1$ and $R_{k+1} > R_k$. Furthermore, by Proposition 3.3, the check whether there is an $X \in \mathcal{F}$ such that $X \supseteq S_1 \cup \{I\}$ and $X \cap S_2 = \emptyset$, can be performed in $O(|\mathcal{H}|)$ by checking if

$$\cup\{H \in \mathcal{H} \setminus (S_1 \cup S_2) \mid L_H > R_k + 1\} \supseteq [R_{k+1} + 1 : \max U]. \tag{3.1}$$

Since the depth of the backtracking tree is at most $|U|$ by Proposition 3.3, the theorem follows. □

3.2. Enumerating minimal cut conjunctions. Let $\mathbf{T} = (V, E)$ be a directed tree with a vertex set V and an arc set E, and let $\mathcal{P} = \{(s_i, t_i) \mid s_i, t_i \in V, \text{ for } i = 1, \ldots, n\}$ be a collection of directed paths in $\mathbf{T}$, defined by source-sink pairs. To apply the generation algorithm described in the previous section, we let $W = E$, and for each $X \subseteq E$, we let π be the property that the graph $(V, E \setminus X)$ has no path between s_i and t_i for $i = 1, \ldots, n$. Clearly, we may assume, without loss of generality, that every leaf of $\mathbf{T}$ is either a source or sink, or both. Let us pick a vertex $r \in V$ arbitrarily to be a root of $\mathbf{T}$, and label all arcs of $\mathbf{T}$ by their *breadth- first search* orders $\{1, \ldots, |E|\}$ from r, in the underlying tree of $\mathbf{T}$.

Lemma 3.4. *Given an arc $i \in W$ and a set $X \in \mathcal{F}_\pi^i$ of i-minimal cut conjunctions such that $X \setminus \{i\}$ is not a cut conjunction, all elements of the family $\mathcal{F}_\pi(i, X)$ can be enumerated with delay $O(|V|)$ and space $O(|V|)$.*

Proof. Since $X \setminus \{i\}$ does not satisfy π, the graph $\big(V, E \setminus (X \setminus \{i\})\big)$ contains a set of paths $\mathcal{P}' \subseteq \mathcal{P}$. Since no such path exists in $(V, E \setminus X)$, each such path must contain arc $i = (a, b)$. Assume without loss of generality that the arc (a, b) points towards r, i.e., b is closer to r than a in the underlying tree of $\mathbf{T}$. Note that the arcs in the subtree of $\mathbf{T}$ rooted at a, (i.e., the arcs that are further from r than (a, b)) are labeled with values higher than i. Since all the paths in $\mathcal{P}'$ avoid all the arcs in $X \setminus \{i\}$ such that $X \supseteq \{i + 1, \ldots, n\}$, none of these arcs appears in the paths in $\mathcal{P}'$. In other words, all the paths in $\mathcal{P}'$ have a common source a, $\mathcal{P}'$ forms an arborescence $\mathbf{T}' = (V', E')$ rooted at a, connecting a to sinks in $\mathcal{P}'$.

The family $\mathcal{F}_\pi(i, X)$ thus consists of all minimal collection of arcs whose removal disconnects a from every sink of $\mathcal{P}'$. We assume without loss of generality that all sink of $\mathcal{P}'$ are leaves in $\mathbf{T}'$, since disconnecting a non-leaf v from a also means disconnecting from a all the nodes in the sub-arborescence of $\mathbf{T}'$ rooted at v. To find the elements of $\mathcal{F}_\pi(i, X)$, we again use a backtracking method that is based on the one described in the previous section.

Let S_1 and S_2 be two disjoint subsets of arcs in E' such that they are extendable to some element of $\mathcal{F}_\pi(i,X)$, i.e., there exists a $Y \in \mathcal{F}_\pi(i,X)$ with $Y \supseteq S_1$ and $Y \cap S_2 = \emptyset$. Let $j \in E' \setminus (S_1 \cup S_2)$. Then it is not difficult to see that $S_1 \cup \{j\}$ and S_2 are extendable to some element of $\mathcal{F}_\pi(i,X)$ if and only if $S_1 \cup \{j\}$ forms an antichain, i.e., there is no directed path in $\mathbf{T}'$ containing two distinct arcs in S_1. Therefore, such an arc j can be found in $O(|V|)$ time. Similarly, S_1 and $S_2 \cup \{j\}$ are extendable to some element of $\mathcal{F}_\pi(i,X)$ if and only if $E' \setminus (S_2 \cup \{j\})$ is a cut conjunction of $\mathcal{P}'$, which can be checked in $O(|V|)$ time. Since the depth of the backtracking tree is at most $|V|$, we have an $O(|V|^2)$ delay algorithm. To reduce the complexity, we modify the algorithm as follows.

Let us first relabel all arcs of $\mathbf{T}'$ by their breadth- first search orders $\{1,\dots,|E'|\}$ from a. In the algorithm, starting from $S_1 = S_2 = \emptyset$, we try to add the least arc j such that $S_1 \cup \{j\}$ and S_2 are extendable to some element of $\mathcal{F}_\pi(i,X)$. Moreover, when we add an arc j to S_1, we add to S_2 all the arcs j' such that j and j' do not form an antichain, i.e., there exists a directed path between j and j', since they are never added to S_1 if $j \in S_1$. Note that by this modification, all the arcs in $E' \setminus (S_1 \cup S_2)$ can be added to S_1, and by the new labeling of arcs, when we add the least arc $j = (c,d)$ to S_1, we just add to S_2 all the arcs in the sub-arborescence of $\mathbf{T}'$ rooted at d. Thus we can go to the left child (and backtrack, i.e., go from the left child to the parent) in $O(\Delta)$ time, where Δ denotes the number of the arcs in the sub-arborescence of $\mathbf{T}'$ rooted at d.

On the other hand, when we add an arc j to S_2, we modify $\mathbf{T}'$ by contracting j. Then we can see that $j = (c,d)$ can be added to S_2 if and only if $c = a$ and d is a leaf of the current $\mathbf{T}'$, and hence we can go to the right child (and backtrack) in $O(1)$ time.

Since the above Δ's are pairwise disjoint in any path in the backtracking tree, the modified algorithm generates the elements of $\mathcal{F}_\pi(i,X)$ with $O(|V|)$ delay and $O(|V|)$ space. □

4. Generating all vertices of a polyhedron is hard

Let us show that problem $\text{Dec}(P,\mathcal{X})$ is NP-complete. Then, as we already mentioned, in view of Proposition 1.1, no algorithm can generate all elements of $\mathcal{V}(P)$ in incremental or total polynomial time, unless P = NP.

Given a directed graph $G = (V,E)$ and a weight function $w\colon E \to \mathbb{R}$ on its arcs, let us define a polyhedron $P(G,w)$ by the following formula:

$$P(G,w) = \left\{ y \in \mathbb{R}^E \;\middle|\; \begin{array}{ll} \displaystyle\sum_{v:(u,v)\in E} y_{uv} - \sum_{w:(w,u)\in E} y_{wu} = 0, & \forall u \in V \\ \displaystyle\sum_{(u,v)\in E} w_{uv} y_{uv} = -1 & \\ 0 \le y_{uv}, & \forall (u,v) \in E \end{array} \right\}.$$

A negative (respectively, positive, or zero) cycle in G is a directed cycle whose total weight is negative (respectively, positive, or zero). Let us denote the families of all negative, positive, and zero-weight cycles of G by $\mathcal{C}^-(G,w)$, $\mathcal{C}^+(G,w)$, and $\mathcal{C}^0(G,w)$, respectively.

Fact 3. For any directed graph $G = (V,E)$ and any real weight $w\colon E \to \mathbb{R}$,

(i) there exists a one-to-one correspondence between $\mathcal{V}\big(P(G,w)\big)$ and $\mathcal{C}^-(G,w)$,

(ii) there exists a one-to-one correspondence between $\mathcal{D}\big(P(G,w)\big)$ and the set $\mathcal{C}^0(G,w) \cup \{(C,C') : C \in \mathcal{C}^-(G,w)$ and $C' \in \mathcal{C}^+(G,w)$ and $C \cup C'$ contains only 2 cycles$\}$.

In other words, the vertices of $P(G,w)$ correspond to the negative cycles of G, whereas the extreme rays correspond to the zero-weight cycles and pairs of negative and positive cycles (see [13]). In the Appendix, we will derive the result of [34] that decision problem $\mathrm{Dec}_{\mathcal{C}^-(G,w)}(G,w,\mathcal{X})$, of checking whether a collection of negative cycles of G is complete, is NP-hard, thus, showing that generating all vertices of $P(G,w)$ is hard.

Theorem 4.1 ([34])**.** *Given a polyhedron P by* (1.1)*, problem* $\mathrm{Gen}(P)$ *of generating all vertices of P is NP-hard.*

However, Fact 3(ii) shows that this construction, given in the Appendix, does not imply the same result for polytops, since the numbers of positive and negative cycles can be exponential and, hence, polyhedron $P(G,w)$ can be highly unbounded.

Appendix A. How to prove that a generation problem is hard?

First, let us notice that there are many generation problems (for instance, generating all Hamiltonian cycles of a given graph) for which already finding the first output object is difficult. However, for a monotone generation problem with a polynomial-time oracle it is always easy to find the first output object or a few objects. Furthermore, whenever we can generate exponentially many output objects then, by the definition of incremental efficiency, the rest of the generation cannot be NP-hard. Hence, the general structure of an NP-hardness proof for a monotone generation problem consists of showing that, after we got polynomially many output objects, deciding the existence of a next one is NP-hard. In this section, we discuss several examples, and conclude with the problem of generating negative cycles of a weighted directed graph. Although the latter problem is not of the monotone type, it is possible to find the first negative cycle in polynomial time, see, e.g. [33]. In the construction described below, we show that after a polynomial number of negative cycles are generated, finding if there is an additional negative cycle is an NP-hard problem.

A.1. Reduction from stability number. As an example for an NP-hard monotone generation problem, let us consider integer programming. Given a system $Ax \geq b$ of m linear inequalities in n integer variables, that is, $A \in \mathbb{R}^{m\times n}$ and $b \in \mathbb{R}^m$, we are looking for integral solutions $x \in \mathbb{Z}^n$ such that $0 \leq x \leq c$, where $c \in \mathbb{R}^n_+$ is a non-negative vector. We get a binary programming problem if all n coordinates of c are equal to 1, that is, $c = \mathbf{1}_n$. It is well-known that in general even the feasibility of such systems is NP-hard to verify. However, if the matrix A is non-negative, $A \geq 0$, then the system is feasible if and only if $Ac \geq b$, and hence the feasibility can be checked in polynomial time. Given such an instance, we consider the problem to generate all integral

(i) minimal feasible and
(ii) maximal infeasible vectors.

It is shown in [11] that (i) can be solved in incremental quasi-polynomial time, while (ii) is NP-hard.

Proposition A.1 ([11]). *Given a system $Ax \geq b$ with $A \geq 0$ and a family $\mathcal{X}$ of its integral maximal infeasible vectors, it is NP-hard to decide whether the family $\mathcal{X}$ is complete or it can be extended. The problem remains NP-hard even if A is a $0/1$-matrix, $c = \mathbf{1}_n$, and all coordinates of b but one are equal to* 1.

Proof. Let us consider the well-known NP-complete decision problem, called *Stability Number*: Given a graph $G = (V, E)$ and a threshold $t \geq 2$, decide if G contains a stable set of size t, or not. Let us introduce $n = |V|$ binary variables x_v, $v \in V$, and write $m - 1 = |E|$ inequalities of the form $x_v + x_{v'} \geq 1$ corresponding to the edges $e = (v, v') \in E$, followed by a single inequality $\sum_{v \in V} x_v \geq n - t$. It is easy to verify that if x is the characteristic vector of an edge $e \in E$ then $\mathbf{1}_n - x$ is a maximal infeasible vector. Furthermore, there is another such vector if and only if G has a stable set of size t. □

A few more generation problems whose hardness is proved by reduction from stability number can be found in [15, 16]. Typically, the input of such a problem contains at least one unbounded parameter. In the above example there is exactly one: the coordinate $n - t$ of vector b.

A.2. Reduction from satisfiability: sausage techniques.

A.2.1. *Reformulations of satisfiability in terms of monotone* DNFs *and* CNFs *and corresponding hard generation problems.* Let $\mathcal{C}$ be a CNF of k Boolean variables $x_1, \ldots, x_k$. It is well-known that verifying the satisfiability of $\mathcal{C}$ is an NP-complete problem that we refer to as *SAT*.

Remark A.1. SAT remains NP-complete even if we assume additionally that

(i) no literal appears in all clauses of $\mathcal{C}$; (*indeed, if x (or $\overline{x}$) appears in all clauses then obviously C is satisfiable.*)

(ii) for every variable x, both literals x and $\overline{x}$ appear in $\mathcal{C}$ (and not in the same clause, of course). (*Indeed, we can substitute $x = 1$, if x appears in $\mathcal{C}$ and $\overline{x}$ does not, and $x = 0$ if $\overline{x}$ appears and x does not. In both cases the obtained* CNF *$\mathcal{C}'$ is satisfiable if and only if $\mathcal{C}$ is satisfiable.*)

It is also known ([50, Theorem 2.1]) *that* SAT *remains NP-hard even if*

(iii) each clause contains at most 3 variables and each variable appears in at most 3 clauses (*i.e., each variable appears in $\mathcal{C}$ once negatively, once positively, and it may appear once more, either negatively or positively*).

A CNF (DNF) is said to be monotone or positive if it has no negated literals. The dual of a monotone CNF (respectively DNF) is the monotone DNF (respectively CNF) obtained by replacing every $\wedge$ with an $\vee$ and vice versa. The *monotone dualization* problem is to find for a given monotone CNF (or DNF) the corresponding monotone DNF (respectively CNF). It is easy to see that this is equivalent to the problem of finding all minimal transversals of a given hypergraph, discussed in Section 1.3.

As an example, the following CNF satisfies all above conditions (i), (ii), and (iii):

$$\mathcal{C} = (x_1 \vee \overline{x}_2 \vee \overline{x}_3)(x_2 \vee x_3)(x_1 \vee \overline{x}_2 \vee x_3)(\overline{x}_1 \vee \overline{x}_2).$$

Given a CNF $\mathcal{C}$, let us assign a positive literal y_i to each negative literal $\overline{x}_i$ of $\mathcal{C}$, denote the obtained monotone CNF by $C = C(\mathcal{C})$ and the dual monotone DNF by $D = D(\mathcal{C})$. For the above example, we get

$$\begin{aligned} C &= (x_1 \vee y_2 \vee y_3)(x_2 \vee x_3)(x_1 \vee y_2 \vee x_3)(y_1 \vee y_2), \\ D &= x_1y_2y_3 \vee x_2x_3 \vee x_1y_2x_3 \vee y_1y_2. \end{aligned} \tag{A.1}$$

Let us also introduce a pair of dual CNF C_0 and DNF D_0 as follows:

$$\begin{aligned} C_0 &= (x_1 \vee y_1) \wedge \cdots \wedge (x_k \vee y_k), \\ D_0 &= x_1y_1 \vee \cdots \vee x_ky_k. \end{aligned} \tag{A.2}$$

Now we can reformulate SAT in many trivially equivalent ways.

Proposition A.2. *The following* 11 *claims are equivalent*:

(0) $\mathcal{C}$ *is not satisfiable,*
(1) $D_0 \geq C$, (2) $D_0 \vee C \equiv D_0$, (3) $D_0 \wedge C \equiv C$,
(4) $C \Rightarrow D_0$, (5) $C\overline{D}_0 \equiv 0$;

(1′) $C_0 \leq D$, (2′) $C_0 \wedge D \equiv C_0$, (3′) $C_0 \vee D \equiv D$,
(4′) $D \Leftarrow C_0$, (5′) $\overline{C} \vee D_0 \equiv 1$.

Proof. Equivalence of (1)–(5) is obvious. Furthermore, (i') is dual to (i) for $i = 1, 2, 3, 4, 5$. It remains to show that (0) and (1) are equivalent. Indeed, both (0) and (1) are obviously equivalent to the following claim: each prime implicant of C contains a pair x_j, y_j for some $j \in [k] = \{1, \ldots, k\}$. □

Remark A.2. Given a monotone DNF $D = t_1 \vee \cdots \vee t_n$ and CNF $C = c_1 \wedge \cdots \wedge c_m$ of common variables $x_1, \ldots, x_k$, inequality $D \geq C$ is NP-hard to verify. In contrast, $D \leq C$ (or equivalently, $D \Rightarrow C$, $D \vee C \equiv C$, $C \wedge D \equiv D$) can be easily verified in linear time. To do so, choose some $i \in [n] = \{1, \ldots, n\}$, set all variables of c_i to 1, and all others to 0. Then, obviously, $D = 1$. It is also clear that $D \not\leq C$ if we get $c_j = 0$ for some $j \in \{1, \ldots, m\}$; otherwise, $C = 1$ whenever $D = 1$, that is, $D \leq C$.

Let us also note that verification of the identity $C \equiv D$ is exactly dualization. As we already mentioned in Section 1.3, this problem can be solved in quasi-polynomial time and, hence, it is not NP-hard unless each problem from NP is quasi-polynomially solvable. Moreover, verifying each of the following dual identities

$$D_1 \vee \cdots \vee D_n \equiv C, \quad C_1 \wedge \cdots \wedge C_n \equiv D,$$

for arbitrary monotone DNFs $D, D_1, \ldots, D_n$ and CNFs $C, C_1, \ldots, C_n$ is obviously equivalent to dualization too. In contrast, verifying each of identities

$$D_1 \wedge \cdots \wedge D_n \equiv C_0, \quad C_1 \vee \cdots \vee C_n \equiv D_0,$$

is NP-hard, since they generalize $D \wedge C_0 \equiv C_0$ and $C \vee D_0 \equiv D_0$, respectively.

Finally, let us note that $C\overline{D}_0$ is equal to a "dipole" CNF, where all clauses of C are positive (contain no negations) and all clauses of $\overline{D}_0 = (\overline{x}_1 \vee \overline{y}_1) \cdots (\overline{x}_k \vee \overline{y}_k)$ are negative (contains only negative literals). Hence, equivalence of claims (0) and (5) implies that SAT is NP-complete already for dipole CNFs, or in other words, that Monotone SAT is NP-complete, a fact that we already made use of in Section 2.3.4.

Some statements from Proposition A.2 can be naturally reformulated as (NP-hard) generation problems. For example, (2) $D_0 \vee C \equiv D_0$, calls for enumerating all prime implicants of a monotone Boolean expression $D_0 \vee C$. One can immediately get k of them, $x_i y_i$ for $i = 1, \dots k$, however, it is NP-hard to decide whether the obtained list is complete or it can be extended. In the next sections we will develop this approach and derive corollaries for reliability theory and vertex enumeration.

Now, let us consider two similar dual identities

$$D_1 \wedge \cdots \wedge D_n \equiv D_0, \quad C_1 \vee \cdots \vee C_n \equiv C_0 \tag{A.3}$$

where D_0 and C_0 are defined by (A.2).

Proposition A.3. *It is coNP-complete to verify any of the identities of* (A.3) *already for monotone quadratic* DNFs $D_1, \dots, D_n$ *and* CNFs $C_1, \dots, C_n$.

Proof. By duality, it will suffice to prove the first claim only. We will reduce it from SAT. Let $\mathcal{C}$ be a CNF (satisfying conditions (i), (ii), and (iii) of Remark A.1) of k variables $x_1, \dots, x_k$. Let us substitute y_j for each $\overline{x}_j$ and get a monotone CNF $C = c_1 \wedge \cdots \wedge c_n$.

Now for $i = 1, \dots, n$ let us set in (A.3) $D_i = D_0 \vee c_i$. Conventionally, we delete the term $x_j y_j$ from D_i if clauses c_i contains x_j or y_j, for $i = 1, \dots, n$ and $j = 1, \dots, m$. Then, by assumption (i), we have $D_1 \wedge \cdots \wedge D_n = D_0$ if and only if the CNF $\mathcal{C}$ is not satisfiable. $\square$

Let us remark that the following, more general, identities

$$D_1 \wedge \cdots \wedge D_n \equiv D, \quad C_1 \vee \cdots \vee C_n \equiv C \tag{A.4}$$

can be reduced to dualization, and hence, verified in quasi-polynomial time if one of the following conditions is satisfied.

Case 1. All DNFs D_i (CNFs C_i) are linear, or in other words, they are just clauses $D_i = c_i = x_i^1 \vee \cdots \vee x_i^{k_i}$ (resp., terms $C_i = t_i = x_i^1 \cdots x_i^{k_i}$) for $i = 1, \dots, n$. In this case (A.4) is reduced to the form $D \equiv C$ which is exactly dualization.

Case 2. The number of terms (resp., clauses) in each of the DNFs $D_1, \dots, D_n$ (resp., CNFs $C_1, \dots, C_n$) is bounded by a constant. In this case each DNF D_i (resp., CNF C_i) can be efficiently dualized in polynomial time, and the size of each such dual is bounded by a polynomial.

We can reformulate Proposition A.3 as NP-hardness of the following generation problem: *Product of Hypergraphs.* To each monotone DNF D let us assign (as usual) a hypergraph $H = (V, E)$ whose vertices are the variables and whose edges are the terms of D.

Proposition A.4. *Given n hypergraphs $H_i = (V, E_i)$, the problem of generating all minimal subsets of V which contain an edge of H_i for each $i = 1, \dots, n$ is NP-hard, even if all H_i are graphs.*

Proof. We can just translate the previous proof (of Proposition A.3) in terms of graphs. Assign a vertex u_j (resp., v_j) to each literal x_j (resp., y_j) for $j = 1, \dots, k$, introduce a new vertex w, and set $V = \{w, u_1, v_1, \dots, u_k, v_k\}$. Then for $i = 1, \dots, n$, let us assign to quadratic DNF $D_i = D_0 \vee c_i$ the graph $G_i = (V, E_i)$ such that $E_i = \{(u_j, v_j) \mid j = 1, \dots, k\} \cup \{(w, u_j) \mid x_j \in c_i\} \cup \{(w, v_j) \mid y_j \in c_i\}$. By Proposition A.3, the product of the obtained n graphs $G_1, \dots, G_n$, or in other words, all minimal subsets of V which contain an edge from E_i for each $i = 1, \ldots, n$, is NP-hard to generate. $\square$

Another corollary of Propositions A.3 and A.4 is also related to the sets of n graphs in which, however, edges, not vertices, are in common. Given n (directed) graphs $G_i = (V_i, E_i)$ for $i = 1, \ldots, n$, whose edges are labeled by the same indices $E = \{e_1, \ldots, e_m\}$, generate all minimal subsets of E such that the corresponding edges contain a (directed) cycle in G_i for each $i = 1, \ldots, n$. It is easy to see that, by Propositions A.3, A.4, this problem is NP-hard. Let us remark that the problem is open for the case when n is bounded by a constant, in particular, for $n = 2$.

We can fix a pair of vertices $s_i, t_i \in V_i$ for each $i = 1, \ldots, n$ and consider (directed) (s_i, t_i) paths instead of (directed) cycles. The corresponding generation problem remains NP-hard.

There are also interesting corollaries of Propositions A.3 and A.4 in reliability theory. For instance, given a directed graph $G = (V, E)$ and n pairs of terminals $s_i, t_i \in V$ for $i = 1, \ldots, n$, consider the problem of generating all minimal subsets of E that contain a directed path from s_i to t_i for each $i = 1, \ldots, n$. It follows from Proposition A.3 that this generating problem is NP-hard [12].

A.2.2. *Sausage technique.* Given a CNF $\mathcal{C}$, let us assign distinct positive variables to all its literals and denote the obtained read-once monotone CNF by C' and the dual read-once monotone DNF by D'. We will denote by $x_i^1, x_i^2, \ldots$ and $y_i^1, y_i^2, \ldots$ the variables of C' and D' corresponding respectively to positive x_i and negative $\overline{x}_i$ literals of $\mathcal{C}$. For our example from Section A.2.1 we get

$$C' = (x_1^1 \vee y_2^1 \vee y_3^1)(x_2^1 \vee x_3^1)(x_1^2 \vee x_3^2)(y_1^1 \vee y_2^2),$$
$$D' = x_1^1 y_2^1 y_3^1 \vee x_2^1 x_3^1 \vee x_1^2 x_3^2 \vee y_1^1 y_2^2.$$

Let us now introduce formulae C_0' and D_0' as follows,

$$C_0' = \bigwedge_{i=1}^{m} \big((x_i^1 x_i^2 \cdots) \vee (y_1^1 y_1^2 \cdots) \big),$$
$$D_0' = \bigvee_{i=1}^{m} \big((x_i^1 \vee x_i^2 \vee \cdots)(y_i^1 \vee y_i^2 \cdots) \big).$$

Let us remark that C_0' and D_0' are dual $(\vee, \wedge)$-formulae of depth 3 rather than a CNF and DNF. For our example we get

$$C_0' = (x_1^1 x_1^2 \vee y_1^1)(x_2^1 \vee y_2^1 y_2^2)(x_3^1 x_3^2 \vee y_3^1),$$
$$D_0' = (x_1^1 \vee x_1^2) y_1^1 \vee x_2^1 (y_2^1 \vee y_2^2) \vee (x_3^1 \vee x_3^2) y_3^1.$$

Again, it is not difficult to see that the inequality $C' \le D_0'$ holds if and only if the original CNF $\mathcal{C}$ is not satisfiable. Furthermore, as in Proposition A.2, we can rewrite $C' \le D_0'$ in several obviously equivalent ways:

$$C' \vee D_0' \equiv D_0', \quad C' \wedge D_0' \equiv C', \quad C' \le D_0', \quad C' \Rightarrow D_0';$$
$$D' \wedge C_0' \equiv C_0', \quad D' \vee C_0' \equiv D', \quad D' \ge C_0'; \quad D' \Leftarrow C_0'.$$

Given a CNF $\mathcal{C}$, let us standardly assign series-parallel graphs $G(C')$, $G(D')$, $G(C_0')$, and $G(D_0')$ to the monotone $(\vee, \wedge)$ Boolean formulae C', D', C_0', and D_0', respectively. Each of these four graphs has two terminals s and t. Let us also note that all four have a common edge-set. By construction, all four are series-parallel and, in particular, planar. Moreover, $\big(G(C'), G(D')\big)$ and $\big(G(C_0'), G(D_0')\big)$ form two pairs of dual planar graphs. (More precisely, they will if we add the edge (s,t) to each one.) Figure 1 shows all four graphs corresponding to the CNF $\mathcal{C}$ considered above.

We can reformulate SAT, as before, in many trivially equivalent ways.

Proposition A.5. *The following* 5 *statements are equivalent*:

(0) CNF $\mathcal{C}$ *is satisfiable.*
(1) *Graph* $G(C_0')$ *has an* s,t*-path whose edge-set contains the edge-set of an* s,t*-path in* $G(C')$.
(1') *There are two edge-disjoint* s,t*-paths*: *one in* $G(C')$, *another in* $G(C_0')$.
(2) *Graph* $G(C_0')$ *has an* s,t*-path whose edge-set contains the edge-set of no* s,t*-path in* $G(D')$.
(2') *Graph* $G(C_0')$ *has an* s,t*-path whose edge-set intersects the edge-set of every* s,t*-path in* $G(D')$.

Proof. Let us show that claims (0) and (1) are equivalent. Indeed, it is easy to see that each s,t-path p_0 in $G(C_0')$ is an assignment of variables of $\mathcal{C}$ and this assignment is satisfying iff the edge-set of p_0 contains the edge-set of some s,t-path p in $G(C)$. Hence, (1) means exactly that there is a satisfying assignment for $\mathcal{C}$.

Now let us note that for each s,t-path in $G(C_0')$ there exists another s,t-path such that the corresponding two edge-sets are complementary.

Moreover, the s,t-paths in $G(C')$ (respectively, in $G(C_0')$) are in a one-to-one correspondence with the s,t-cuts in $G(D')$ (respectively, in $G(D_0')$), since $\big(G(C'), G(D')\big)$ and $\big(G(C_0'), G(D_0')\big)$ form two pairs of dual planar graphs.

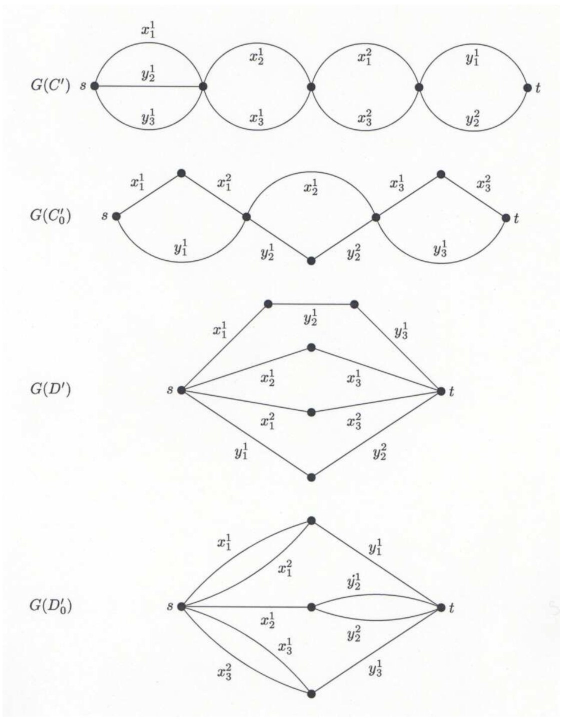

Figure 1. Four labeled graphs corresponding to the CNF $\mathcal{C}$.

These two observations easily imply that all 5 above claims are equivalent. □

Furthermore, we can substantially extend the list substituting a path of a graph by the corresponding cut of the dual graph.

Since SAT is NP-complete, we get a long list of NP-complete problems related to pairs of graphs $G' = (V', E)$, $G'' = (V'', E)$ with common edge-sets.

In particular, it is NP-hard to check if G' and G'' have edge-disjoint (or edge nested) s, t-paths (or s, t-cuts); whether they have edge-disjoint (or edge-nested) pair of an s, t-path and s, t-cut; whether G' has an s, t-cut (or s, t-path) whose edge set contains no s, t-path (or s, t-cut, or vice versa) of G'', etc.

Remark A.3. All the problems above remain NP-complete for directed graphs too. To get a proof it is sufficient to orient all edges of all four graphs in direction from s to t. It is well known that for series-parallel graphs (and, in fact, only for them) this operation is well-defined.

Finally, let us note that each of the above problems still remains NP-complete if we substitute simple (directed) cycles for (directed) s, t-paths. Indeed, let us identify the terminals s and t in $G(C')$ and in $G(C'_0)$. Then in these two (directed) graphs each (directed) s, t-path turns into a simple (directed) cycle. We should note that:

(a) the number of s, t-paths may be exponential in size of CNF $\mathcal{C}$ for $G(C')$ and $G(C'_0)$, while for $G(D')$ and $G(D'_0)$ it is at most linear and, respectively, quadratic in size of $\mathcal{C}$;

(b) in all four graphs there are other simple (directed) cycles, not related to (directed) s, t-paths. However, their number is at most linear in the size of $\mathcal{C}$. Hence, they can be checked separately and cannot influence the complexity.

We showed that Proposition A.5 leads to many NP-hard decision problems. Let us now demonstrate that they can be easily reformulated as generation problems. The following statement explains the method.

Given two (directed) graphs $G' = (V', E)$ and $G'' = (V'', E)$ with a common edge-set E, generate all minimal subsets of E that contain a simple (directed) cycle in G' or in G''. This problem is called *disjunction of cycles* [37].

Proposition A.6. *Disjunction of cycles is an NP-hard generation problem.*

Proof. Consider the non-directed case first. Let us merge vertices s and t in $G(D')$ and $G(C'_0)$ and denote the obtained graphs by G' and G'', respectively. Then we can easily enumerate all simple cycles of G' and all "short" simple cycles of G''. However, by Proposition A.5 it is NP-complete to decide if there is a "long" simple cycle in G'' whose edge-set contains no edge-set of a simple cycle in G'.

Exactly the same arguments work in the directed case too. We only have to orient all edges of $G(D')$ and $G(C'_0)$ from s to t. Let us note that in this case there are no "short" simple cycles in G''. □

Remark A.4. Clearly, the similar statement holds if we substitute (directed) s, t-paths or s, t-cuts for simple (directed) cycles.

In the previous subsection we considered the similar concept of *conjunction of* (*directed*) *s, t-paths or simple cycles* in n graphs. Let us recall that the corresponding generation problem is NP-hard when n is a part of the input, and it is open when n is bounded, already for $n = 2$.

We refer to the method of this subsection as "*sausage technique*," because the graphs $G(C')$ and $G(C'_0)$ look like a sausage.

A.2.3. *Generating all negative directed cycles is hard.* Given a directed graph $G = (V, E)$ and a real weight function $w\colon E \to \mathbb{R}$ on its edges, consider the problem of generating all negative directed cycles, or more precisely, all simple directed cycles C in G such that $\sum_{e\in E(C)} w(e) < 0$. This generation problem turns out to be NP-hard.

Proposition A.7. *Given G, w and a collection of negative cycles $\mathcal{X}$, it is NP-hard to decide whether this collection is complete or it can be extended. The problem remains NP-hard, even if w takes only two values: $+1$ and -1.*

In [34] this result is proved for both directed and non-directed graphs. Here we give a sketch of this proof for the directed case. Graph G ("double sausage") is constructed as follows.

Let us merge vertex t of graph $G(C')$ and vertex s of graph $G(C'_0)$ and denote the obtained vertex by u_0. The obtained graph G_1 is still series-parallel, so let us orient all its edges from s in $G(C')$ towards t in $G(C'_0)$.

Let us recall that graphs $G(C')$ and $G(C'_0)$ had the common edge-set E which is in one-to-one correspondence with the literals of the CNF $\mathcal{C}$. Hence, the edges of the obtained directed graph G_2 are labeled by E and each label $e \in E$ appears exactly twice.

Furthermore, it is NP-hard to check if G_2 contains an s,t-path in which each label $e \in E$ appears at most once. Indeed, such a path exists in G_2 if and only if there exist two edge-disjoint s,t-paths in $G(C')$ and $G(C'_0)$, which is NP-hard to verify by Proposition A.5.

Now let us subdivide each directed edge (v', v'') in G_2 by 3 vertices v_1, v_2, v_3 such that v', v_1, v_2, v_3, v'' go successively. In the obtained directed graph G_3 let us define weight function w as follows: $w(v', v_1) = w(v_3, v'') = +1$, $w(v_1, v_2) = w(v_2, v_3) = -1$.

Furthermore, for each index $e \in E$ let us consider two corresponding directed edges (v', v'') and (u', u'') in G_2 and their subdivisions v', v_1, v_2, v_3, v'' and u', u_1, u_2, u_3, u'' in G_3; then let us merge v_1 and u_3 and also v_3 and u_1, and denote the two obtained vertices by v and u, respectively; see Figure 2. Finally, let us add one more directed edge (t, s) and assign to it weight -1.

By construction, vertices v, v_2, u, u_2 form a simple directed cycle of weight -4 in the obtained directed graph G_4. There are $|E|$ such cycles. Does G_4 contain more negative cycles? We will show that this question is NP-hard.

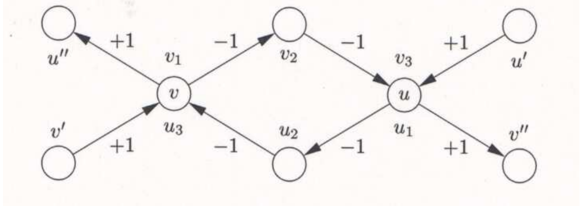

FIGURE 2. Two pairs of vertices are merges.

Let us note that there is a directed cycle of weight -1 in G_4 whenever there exist two edge-disjoint s,t-paths in graphs $G(C')$ and $G(C'_0)$ (and this is NP-hard to decide, by Proposition A.5). Indeed, the union of these two paths is an s,t-path in G_4 whose total weight is 0. Hence, these two paths and the edge (t,s) form a simple directed cycle in G_4 of weight -1.

It is not difficult to verify that there can be no other negative cycles in G_4. In other words, G_4 contains $|E|$ (-4)-cycles and the projection of any other negative cycle Y to $G(C')$ and $G(C'_0)$ must form an s,t-path in each graph. The complete proof is given in [34]. However, its main point is simple. If Y contains successive vertices u_2, v, v_2 or v_2, u, u_2 then Y is a (-4)-cycle. If Y contains successive vertices v', v, u'' or u', u, v'' then Y cannot be negative, moreover, $w(Y) \geq 1$.

References

1. S. D. Abdullahi, M. E. Dyer, and L. G. Proll, *Listing vertices of polyhedra associated with dual* LI(2) *system*, Discrete Mathematics and Theoretical Computer Science (Dijon), Lecture Notes in Comput. Sci., vol. 2731, Springer, Berlin, 2003, pp. 89–96.
2. D. Avis and K. Fukuda, *A pivoting algorithm for convex hulls and vertex enumeration of arrangements and polyhedra*, ACM Symposium on Computational Geometry (North Conway, 1991), Discrete Comput. Geom. **8** (1992), no. 3, 295–313.
3. ———, *Reverse search for enumeration*, Discrete Appl. Math. **65** (1996), 21–46.
4. C. Berge, *Hypergraphs*, North Holland Mathematical Library, vol. 445, Elsevier-North Holland, Amsterdam, 1989.
5. J. C. Bioch and T. Ibaraki, *Complexity of identification and dualization of positive Boolean functions*, Inform. and Comput. **123** (1995), 50–63.
6. E. Boros, K. Elbassioni, and V. Gurvich, *Algorithms for generating minimal blockers of perfect matchings in bipartite graphs and related problems*, Algorithms—ESA 2004 (Bergen), Lecture Notes in Comput. Sci., vol. 3221, Springer, Berlin, 2004, pp. 122–133.
7. E. Boros, K. Elbassioni, V. Gurvich, and L. Khachiyan, *An efficient incremental algorithm for generating all maximal independent sets in hypergraphs of bounded dimension*, Parallel Process. Lett. **10** (2000), 253–266.
8. ———, *Generating dual-bounded hypergraphs*, Optim. Methods Softw. **17** (2002), 749–781.
9. ———, *Generating maximal independent sets for hypergraphs with bounded edge-intersections*, Theoretical Informatics, (Buenos Aires), Lecture Notes in Comput. Sci., vol 2976, Springer, Berlin, 2004, pp. 488–498.
10. ———, *Enumerating minimal dicuts and strongly connected subgraphs and related geometric problems*, Integer Programming and Combinatorial Optimization (New York), Lecture Notes in Comput. Sci., vol. 3064, Springer, Berlin, 2004, pp. 152–162.
11. E. Boros, K. Elbassioni, V. Gurvich, L. Khachiyan, and K. Makino, *Dual-bounded generating problems: all minimal integer solutions of a monotone systems of linear inequalities*, SIAM J. Comput. **31** (2002), 1624–1643.
12. ———, *Generating paths and cuts in multi-pole (di)graphs*, Mathematical Foundations of Computer Science (Prague), Lecture Notes in Computer Science, vol. 3153, Springer, Berlin, 2004, pp. 298-309; Disc. Appl. Math. (to appear)
13. E. Boros, K. Elbassioni, V. Gurvich, and H. R. Tiwary, *The negative cycles polyhedron and hardness of checking some polyhedral properties*, RUTCOR Research Report RRR 5–2008, Rutgers University.
14. E. Boros, V. Gurvich, and P. L. Hammer, *Dual subimplicants of positive Boolean functions*, Optim. Methods Softw. **10** (1998), 147–156.
15. E. Boros, V. Gurvich, L. Khachiyan, and K. Makino, *Dual-bounded generating problems: weighted transversals of a hypergraph*, Discrete Appl. Math. **142** (2004), 1–15.
16. ———, *Dual-bounded generating problems: partial and multiple transversals of a hypergraph*, SIAM J. Comput. **30** (2001), 2036–2050.
17. D. Bremner, K. Fukuda, and A. Marzetta, *Primal-dual methods for vertex and facet enumeration*, Discrete Comput. Geom. **20** (1998), 333–357.

18. M. Bussieck and M. Lübbecke, *The vertex set of a 0/1-polytope is strongly P-enumerable*, Comput. Geom. **11** (1998), 103–109.
19. R. Collado, A. Kelmans, and D. Krasner, *On convex polytopes in the plane "containing" and "avoiding" zero*, DIMACS Tech. Report 2002–33, Rutgers University.
20. C. Domingo, N. Mishra, and L. Pitt, *Efficient read-restricted monotone CNF/DNF dualization by learning with membership queries*, Machine Learning **37** (1999), 89–110.
21. T. Eiter and G. Gottlob, *Identifying the minimal transversals of a hypergraph and related problems*, SIAM J. Comput. **24** (1995), 1278–1304.
22. T. Eiter, G. Gottlob, and K. Makino, *New results on monotone dualization and generating hypergraph transversals*, SIAM J. Comput. **32** (2003), 514–537.
23. T. Eiter, K. Makino, and G. Gottlob, *Computational aspects of monotone dualization: a brief survey*, KBS Research Report INFSYS RR–1843–06–01, Vienna University of Technology Favoritenstraße 9–11, 2006.
24. M. L. Fredman and L. Khachiyan, *On the complexity of dualization of monotone disjunctive normal forms*, J. Algorithms **21** (1996), 618–628.
25. K. Fukuda and T. Matsui, *Finding all minimum-cost perfect matchings in bipartite graphs*, Networks **22** (1992), 461–468.
26. ———, *Finding all the perfect matchings in bipartite graphs*, Appl. Math. Lett. **7** (1994), 15–18.
27. R. M. Freund and J. B. Orlin, *On the complexity of four polyhedral set containment problems*, Math. Programming **33** (1985), 139–145.
28. M. R. Garey and D. S. Johnson, *Computers and intractability, a guide to the theory of NP-completeness*, W. H. Freeman and Company, San Francisco, 1991.
29. J. Gleeson and J. Ryan, *Identifying minimally infeasible subsystems of inequalities*, ORSA J. Comput. **2** (1990), 61–63.
30. V. Gurvich and L. Khachiyan, *On generating the irredundant conjunctive and disjunctive normal forms of monotone Boolean functions*, Discrete Appl. Math. **1996–97** (1999), 363–373.
31. D. S. Johnson, *Open and closed problems in NP-completeness*, NP-Completeness: The First 20 Years (Erice, 1991).
32. D. S. Johnson, M. Yannakakis, and C. H. Papadimitriou, *On generating all maximal independent sets*, Inform. Process. Lett. **27** (1988), 119–123.
33. R. Karp, *A characterization of the minimum cycle mean in a digraph*, Discrete Math. **23** (1978), 309–311.
34. L. Khachiyan, E. Boros, K. Borys, K. Elbassioni, and V. Gurvich, *Generating all vertices of a polyhedron is hard*, Proc. of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithms (Miami, FL, 2006), ACM Press, 2006, pp. 758–765.
35. L. Khachiyan, E. Boros, K. Borys, K. Elbassioni, V. Gurvich, and K. Makino, *Generating cut conjunctions and bridge avoiding extensions in graphs*, Algorithms and Computation (Hainan), Lecture Notes in Comput. Sci., vol. 3827, Springer, Berlin, 2005, pp. 156–165.
36. ———, *Enumerating spanning and connected subsets in graphs and matroids*, Algorithms (Zurich), Lecture Notes in Comput. Sci., vol. 4168, Springer, Berlin, 2006, pp. 444–455.
37. L. Khachiyan, E. Boros, K. Elbassioni, V. Gurvich, and K. Makino, *On the complexity of some enumeration problems for matroids*, SIAM J. Discrete Math. **19** (2005), 966–984.
38. A. Lehman, *On the width-length inequality*, Mimeographic Notes, 1965; Math. Programming **17** (1979), 403–417.
39. E. Lawler, J. K. Lenstra and A. H. G. Rinnooy Kan, *Generating all maximal independent sets: NP-hardness and polynomial-time algorithms*, SIAM J. Comput. **9** (1980), 558–565.
40. L. Lovász, *Combinatorial optimization: some problems and trends*, DIMACS Tech. Report 1992-53, Rutgers University, 1992.
41. L. Lovász and M. D. Plummer, *Matching theory*, North-Holland, Amsterdam, 1986.
42. K. Makino, *Efficient dualization of $O(\log n)$-term monotone disjunctive normal forms*, Discrete Appl. Math. **126** (2003), 305–312.
43. K. Makino and T. Ibaraki, *The maximum latency and identification of positive Boolean functions*, SIAM J. Comput. **26** (1997), 1363–1383.
44. C. Papadimitriou, *NP-completeness: a retrospective*, Automata, Languages, and Programming, Lecture Notes in Comput. Sci., vol. 1256, Springer, Berlin, 1997, pp. 2–6.

45. M. E. Pfetsch, *The maximum feasible subsystem problem and vertex-vacet incidences of polyhedra*, Dissertation, TU Berlin, 2002.
46. Septembre J. S. Provan, *Efficient enumeration of the vertices of polyhedra associated with network* LP*'s*, Math. Programming **64** (1994), 47–64.
47. R. C. Read and R. E. Tarjan, *Bounds on backtrack algorithms for listing cycles, paths, and spanning trees*, Networks **5** (1975), 237–252.
48. A. Schrijver, *Theory of linear and integer programming*, Wiley-Interscience, Chichester, 1986.
49. B. Schwikowski and E. Speckenmeyer, *On enumerating all minimal solutions of feedback problems*, Discrete Appl. Math. **117** (2002), 253–265.
50. C. A. Tovey, *A simplified NP-complete satisfiability problem*, Discrete Appl. Math. **8** (1984), 85–89.
51. S. Tsukiyama, M. Ide, H. Ariyoshi, and I. Shirakawa, *A new algorithm for generating all maximal independent sets*, SIAM J. Comput. **6** (1977), 505–517.
52. T. Uno, *Algorithms for enumerating all perfect, maximum and maximal matchings in bipartite graphs*, Algorithms and Computation (Singapore), Lecture Notes in Comput. Sci., vol. 1350, Springer, Berlin, 1997, pp. 92–101.
53. ———, *A fast algorithm for enumerating bipartite perfect matchings*, Algorithms and Computation (Christchurch), Lecture Notes in Comput. Sci., vol. 2223, Springer, Berlin, 2001, pp. 367–379.
54. G. M. Ziegler, *Lectures on polytopes*, Grad. Texts in Math. **152**, Springer, Berlin, 1995.

RUTCOR, Rutgers University, 640 Bartholomew Road, Piscataway, NJ 08854-8003, USA
E-mail address: `boros@rutcor.rutgers.edu`

Max-Planck-Institut für Informatik, Saarbrücken, Germany
E-mail address: `elbassio@mpi-sb.mpg.de`

RUTCOR, Rutgers University, 640 Bartholomew Road, Piscataway, NJ 08854-8003, USA
E-mail address: `gurvich@rutcor.rutgers.edu`

Graduate School of Information Science and Technology, University of Tokyo, Tokyo, 113-8656, Japan
E-mail address: `makino@mist.i.u-tokyo.ac.jp`