University of Economics, Prague
Faculty of Informatics and Statistics
Department of Econometrics

Technical Report 01/16

A new algorithm for enumeration of cells of hyperplane arrangements and a comparison with Avis-Fukuda's Reverse Search

Miroslav Rada and Michal Černý

Preprint

October 2016

A NEW ALGORITHM FOR ENUMERATION OF CELLS OF HYPERPLANE ARRANGEMENTS AND A COMPARISON WITH AVIS-FUKUDA'S REVERSE SEARCH

MIROSLAV RADA* AND MICHAL ČERNÝ†

Abstract. We design a new algorithm, called Incremental Enumeration (IncEnu), for enumeration of full-dimensional cells of hyperplane arrangements (or dually, for enumeration of vertices of generator-presented zonotopes). The algorithm is based on an incremental construction of the graph of cells of the arrangement.

IncEnu is compared to Avis-Fukuda's Reverse Search (RS), including its later improvements by Sleumer and others. The basic versions of IncEnu and RS are directly incomparable since they solve different numbers of linear programs (LPs) of different sizes. Thus we reformulate our algorithm into versions that allow for a comparison in terms of numbers of LPs solved. The result is that both IncEnu and RS have "the same" complexity-theoretic properties (compactness, output-polynomiality, worst-case bounds, tightness of the bounds). In spite of the same asymptotic bound, it is proved that IncEnu is faster than RS by an interesting additive term.

We also perform computational experiments. For most of our test cases, IncEnu is significantly faster than RS from the practical viewpoint. Based on the results, we conjecture that IncEnu is $\mathcal{O}(d)$ times faster for non-degenerate arrangements, where d denotes the dimension of the arrangement.

Key words. Arrangements, Cell enumeration, Zonotopes, Output-sensitive algorithm, Compact algorithm

AMS subject classifications. 52C35

1. Introduction. We design a new algorithm, called *Incremental Enumeration Algorithm* (IncEnu, IE), for enumeration of (full-dimensional) cells of an arrangement of m hyperplanes in $\mathbb{R}^d$, or dually, for enumeration of vertices of a zonotope in $\mathbb{R}^d$ given by m generators. The algorithm requires polynomial computation time per cell ("output-polynomiality") and polynomial memory in the size of input ("compactness"). Recall that the number of cells can be [3, 5, 10]

$$\zeta(m,d) := \sum_{i=0}^{d} \binom{m}{i}, \tag{1.1}$$

which is superpolynomial in m in general, e.g. if $d = m/2$.[1] Thus, output-polynomiality is "the best" what can be achieved. Compactness means that so-far-produced output cannot be stored in memory; the output must be printed as a stream.

Among cell enumeration algorithms, Avis-Fukuda's Reverse Search (RS) [2] can be considered as a standard. Recall that it is also an output-polynomial and compact algorithm.

For both approaches, IE and RS, the most time-consuming subprocedure is linear programming (LP). Unfortunately, IE and RS are not directly comparable since they

*University of Economics, Prague, Faculty of Finance and Accounting, nám. W. Churchilla 4, 130 67 Prague 3, Czech Republic. (miroslav.rada@vse.cz) The work was supported by by GA ČR P403/16-00408S. Also, it was supported by Institutional Support of University of Economics, Prague, VŠE IP 100040.

†University of Economics, Prague, Faculty of Informatics and Statistics, nám. W. Churchilla 4, 130 67 Prague 3, Czech Republic. (cernym@vse.cz) The work was supported by GA ČR P403/16-00408S.

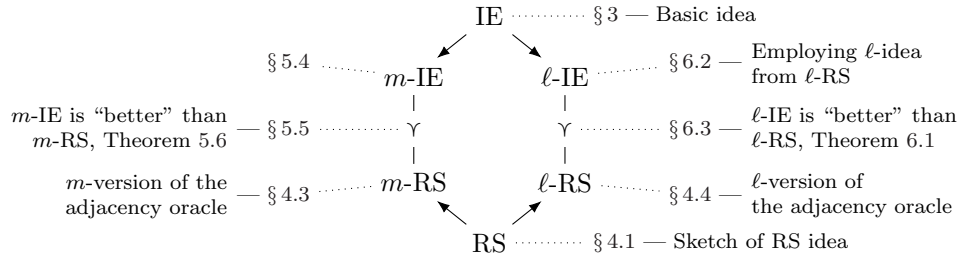
[1] Observe that $\zeta(m,d)$ is polynomial if $d = \mathcal{O}(1)$; this special case has important applications in quadratic programming [1, 6].

solve different numbers of LPs of different sizes. Most work of the paper is devoted to designing a method allowing for a comparison between IE and RS.

RS is currently known in (at least) two versions, called m-RS [6] and ℓ-RS [9], which differ by the method of listing neighbour cells of a given cell (known as *adjacency oracle*). To be able to compare IE and RS, we design two versions of IE, called m-IE and ℓ-IE, which can be compared to the respective versions of RS. The main results are stated in Theorems 5.6 and 6.1, showing that $\mathfrak{r}$-IE is "better" than $\mathfrak{r}$-RS for both $\mathfrak{r} \in \{m, \ell\}$. In fact, we conclude that *whichever version of Reverse Search one would use, the corresponding version of IncEnu performs better in terms of time.*

Aside from RS algorithm, there are several other algorithms that are notable in context of cell enumeration for arrangements. First, there is the asymptotically optimal algorithm for enumerating *all* faces of a hyperplane arrangement by Edelsbrunner et al. [5]. However, this algorithm is not compact. Second, there is a quite significant resemblance of our algorithm with the vertex enumeration algorithm of Bussieck and Lübbecke [4], whose formulation uses the same structure of the main recursive procedure as our algorithm. Their algorithm works on polytopes, whose vertices are among the vertices of a polytope combinatorially equivalent to hypercube. Zonotopes satisfy this property, however, the algorithm in the basic form expects a halfspace description on the input. The obtaining of halfspace description is not computationally feasible for zonotopes. It would be nice to find an explicit interconnection between both the algorithms.

The structure of the paper is depicted in the following scheme.



To illustrate the practical behavior, in § 7 we present computational experiments comparing IE, m-IE, ℓ-IE, m-RS and ℓ-RS. The experiments show that IE is the fastest one.

2. Preliminaries.

Cells and sign vectors. Let a family of hyperplanes $\{h_i := \{x \mid g_i^T x = 1\},\ i = 1, \dots, m\}$ in $\mathbb{R}^d$ be given. The system of (full-dimensional) cells of the hyperplane arrangement spanned by the hyperplanes is denoted by $\mathcal{C}$. (The unit right-hand sides of $g_i^T x = 1$ are without loss of generality when the task is to enumerate cells.)

The matrix with columns $g_1, \dots, g_m$ is assumed to be of rank d. A cell $C \in \mathcal{C}$ can be identified with a *sign vector* $s \in \{\pm 1\}^m$ if $s_i(g_i^T \xi - 1) > 0$ for all i, where ξ is an (arbitrary) interior point of C. Observe that only some sign vectors $s \in \{\pm 1\}^m$ determine cells.

If $h_i = \{x \mid g_i^T x = 1\}$, we write $h_i^+ = \{x \mid g_i^T x \geq 1\}$ and $h_i^- = \{x \mid g_i^T x \leq 1\}$.

Neighbours and flips. Let s, s' be sign vectors differing in exactly one sign, say i-th. We say that s' results from s by *plus-to-minus flip*, or *PM-flip* for short, if $s_i = 1$ and $s'_i = -1$. Similarly, s' results from s by *MP-flip* if $s_i = -1$ and $s'_i = 1$. In both cases we write $s' = \mathrm{flip}_i(s)$. If both s, s' generate cells, then the cells are called *neighbour* and the (unique) hyperplane separating them is called *border hyperplane.*

Assume s generates a cell. If so does $\mathrm{flip}_i(s)$, we say that the sign flip is *successful*; otherwise it is *unsuccessful*.

Convention. By upper indices we will denote dimensions. In particular, the upper index of $s^i \in \{\pm 1\}^i$ emphasizes that s^i is a sign vector with i signs.

Arrangements $\mathcal{A}_i$, successors, witness points. For $i \le n$, let $\mathcal{A}_i$ denote the arrangement spanned by $\{g_\iota^T x = 1 \mid \iota = 1, \dots, i\}$ and let $\mathcal{C}_i$ denote the system of its cells. Let a sign vector $s^i \in \{\pm 1\}^i$ determine a cell $C(s^i) \in \mathcal{C}_i$. For $j > i$, a sign vector $s^j = \binom{s^i}{t^{j-i}}$ with some $t^{j-i} \in \{\pm 1\}^{j-i}$ is called *successor* of s^i. If, in addition, s^j generates a cell $C(s^j)$ of $\mathcal{A}_j$, the cell $C(s^j)$ is called *successor cell* of $C(s^i)$. If $j = i + 1$, it is an *immediate* successor.

An interior point of a cell C is called *witness*. A witness point for a cell generated by sign vector s^i is denoted by $w(s^i)$. A useful property, which can be w.l.o.g. assumed, is: if $j > i$, then a witness ξ of $C(s^i) \in \mathcal{C}_i$ is also a witness of a successor $C(s^j) \in \mathcal{C}_j$ for some $s^j \in \{\pm 1\}^m$ (the degenerate case $g_\iota^T \xi = 1$ for some $\iota = i + 1, \dots, j$ can be easily resolved by a perturbation of ξ).

The arrangement $\mathcal{A}_0$ is defined by an empty list of hyperplanes and a single cell $C(s^0) := \mathbb{R}^d$.

Convention. Let G_i denote the matrix with columns $g_1, \dots, g_i$. Given a vector z, $\mathrm{diag}(z)$ is the diagonal matrix of z.

Testing whether a given sign vector s^i generates a cell. Finding witnesses. The cell-generating test amounts to solving a single LP and checking its optimal value: $C(s^i) \in \mathcal{C}_i$ iff

$$\max\{\varepsilon \mid \mathrm{diag}(s^i)(G_i\xi - 1) \ge \varepsilon,\ \varepsilon \le 1\} > 0. \tag{2.1}$$

Note that this LP also provides through ξ a witness of $C(s^i) \in \mathcal{C}_i$ (if it exists). This allows us to assume that a cell is associated with a *unique* "central" witness $w_u(\cdot)$. In other words, we assume that we have a fixed (poly-time computable) mapping $s^i \mapsto w_u(s^i)$.

Assume we perform the test (2.1) for a sign vector s^i using LP. If the test passes, we call the LP *successful* (since it helps to find a new cell), otherwise we call it *unsuccessful*.

In complexity statements, we use symbol $\mathrm{lp}(\mu, d)$. It means space- or time-complexity needed to solve linear program with $\mu + \mathcal{O}(1)$ constraints and $d + \mathcal{O}(1)$ variables with data of bit-size bounded by the bit-size of G.

3. The basic algorithm: IncEnu. We present the basic algorithm in two equivalent formulations: IE and FlIE. The reason is that IE is conceptually very simple and easy to believe to be correct, while FlIE is in a form that is quite similar to the RS algorithm, which allows for an easier comparison of both the algorithms.

3.1. The easy version: IE. The basic version of the algorithm IncEnu (IE) can be formulated as the tail-recursive procedure in Algorithm 1. It enumerates all cells of $\mathcal{A}_m$ when run as $\mathrm{IncEnu}(s^0, 0)$, where s^0 is an empty sign vector.

At each recursive call, the algorithm works on a single cell-generating sign vector s^i. It is assumed that its witness point w is known, too.

If $i = m$, then the current s^i generates a cell of $\mathcal{A}_m$ and shall be output.

If $i < m$, then the algorithm adds hyperplane h_{i+1} to the arrangement and examines, how this hyperplane affects the current cell $C(s^i)$: there are three possibilites:

(a) either the whole cell $C(s^i)$ belongs

- either into the halfspace h_{i+1}^+,

{1} **Function** IE(s^i, w):
{2} **if** $i < m$:
{3} **if** $w \in h^+_{i+1}$: $\sigma = 1$ **else**: $\sigma = -1$
{4} IE($\binom{s^i}{\sigma}, w$)
{5} **if** exists witness w' of $C\binom{s^i}{-\sigma}$:
{6} IE($\binom{s^i}{-\sigma}, w'$)
{7} **else**:
{8} **output** s^i

{f1} **Function** FlIE(s^i, w):
{f2} $s^m :=$ fullSized(w)
{f3} **output** s^m
{f4} **for** $j = m, m-1, \ldots, i+1$:
{f5} $s^j :=$ flip$_j$(shorten(s^m, j))
{f6} **if** exists witness w' of $C(s^j)$:
{f7} FlIE(s^j, w')

Algorithm 1: Equivalent formulations of IncEnu: IE and FlIE. The input variables: s^i: a sign vector such that $C(s^i)$ is a cell of $\mathcal{A}_i$, w: its witness point. Note that both the variables can be treated as global.

- or into h^-_{i+1};

then only the immediate successor $\binom{s^i}{\sigma}$ of s^i generates a cell of $\mathcal{A}_{i+1}$ for the appropriate σ (chosen on step {3}) and can serve as the new sign vector, that shall be processed by recursive call of IncEnu,

(b) or h_{i+1} subdivides $C(s^i)$ into two new cells of $\mathcal{A}_{i+1}$; then both the immediate successors of s^i generate cells of $\mathcal{A}_{i+1}$ and shall be consecutively processed by recursive calls of IncEnu.

The cases (a) and (b) can be distinguished using the test (2.1) – making it simply test the existence of witness point for $C\binom{s^i}{-\sigma}$.

Solving the LP in {5} is definitely the most time-consuming task of the IncEnu procedure. It is natural to examine the time complexity in terms of the overall number and sizes of the LPs to be solved.

3.2. FlIE: the version more comparable with RS. In fact, recursive calls of IE on step {4} work as follows: a witness point w of a cell $C(s^i)$ of $\mathcal{A}_i$ is taken and the sign vector s^i is completed to s^m such that w is a witness of $C(s^m)$. The sign vector is being prolonged by one sign per call, the prolonging sign is being chosen on step {3}.

The FlIE algorithm is a reformulation of IE such that it groups the prolongings on step {3} of IE into one call of fullSized(w) (on step {f2}). The recursive calls of IE on step {4} are not necessary in FlIE anymore.

The procedure shorten(s^i, j) with $i > j$ cuts of last $(i-j)$ signs of s^i. The **for** cycle on steps {f4}–{f7} works as steps {5} and {6} in consecutive calls of IE. In fact, it takes s^m and shortens it to the appropriate length in each iteration to obtain corresponding s^i as in the corresponding IE call.

The course of FlIE algorithm can be viewed as *fixing signs*, too. Initially, a witness w of a cell $C(s^m)$ is given and no sign in s^m is fixed. Flip of every sign of s^m is checked during FlIE(s^0, w) call. During FlIE(s^i, w) call, the first i signs of s^m are *fixed* and only signs with indices $i+1, \ldots, m$ are going to be flipped.

3.3. Example run of IE and FlIE. Runs of both IE and FlIE are visualized in Figure 1. Every black edge corresponds to finding one new witness point and a new recursive call of IE (resp. FlIE). Green edges and nodes corresponds to "free" prologings without LP. Note that these are "grouped" in the right picture. Red nodes and edges corresponds to infeasible LPs.

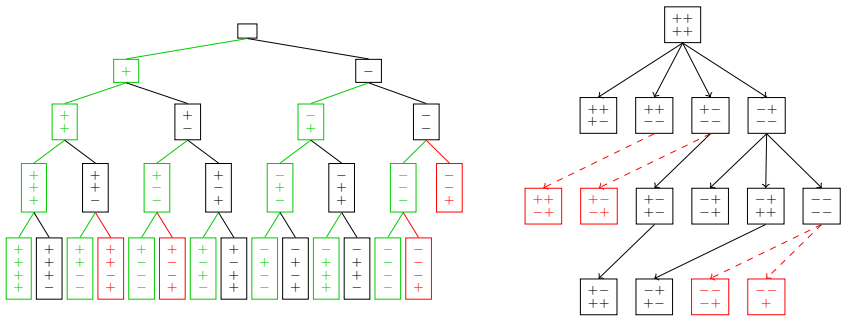


Fig. 1: IE (left tree) and FlIE (right tree) for a sample arrangement with $m = 4$ and $d = 3$. Red nodes: infeasible sign vectors (decided by LP). Green nodes: feasible sign vectors found without LP. Black nodes: feasible sign vectors found with LP.

3.4. Complexity and correctness of IncEnu. The IE version is discussed.

THEOREM 3.1 (Correctness). *Given an arrangement $\mathcal{A}_m$, the IE$(s^0, 0)$ procedure outputs a stream of sign vectors generating cells of $\mathcal{A}_m$ such that the stream is complete and contains no duplicities.*

Proof.

(a) *Uniqueness.* A cell is generated by a unique sign vector. Thus, every call of IE gets a different input sign vector s^i.

(b) *Completeness.* All sign vectors generating cells of an arrangement $\mathcal{A}(G_1)$ are clearly visited by some IE$(s^1, w(s^1))$ calls on the first level of recursion.
Now, assume that all cells of an arrangement $\mathcal{A}(G_i)$ are visited at i-th level of recursion. For each $C(s^i) \in \mathcal{C}_i$ generated by s^i, both successors s^{i+1} of s^i are tested whether they generate a cell of $\mathcal{A}_{i+1}$.
If there exists a cell $C(s^{i+1}) \in \mathcal{C}_{i+1}$ such that it was not examined in this way, we get a contradiction, since s^{i+1} has an immediate predecessor s^i that generates a cell $C(s^i)$. □

THEOREM 3.2 (Complexity).

(a) Time complexity of IE$(s^0, 0)$ is $\mathcal{O}(|\,\mathcal{C}_m|\ m\ \mathrm{lp}(m, d))$.

(b) Space complexity of IE$(s^0, 0)$ is $\mathcal{O}(\mathrm{lp}(m, d))$.

Proof.

In every IE call, either a sign vector s^m generating a cell is output, or at least one another IE procedure is called. During an IE course, at most one linear program with complexity $\mathrm{lp}(G)$ is solved.

To prevent the degenerate case $g_{i+1}^T w = 1$, we also need computation time for perturbation of the witness w. For perturbation one must compute safe bounds to ensure that the perturbed point is still an interior point of the examined cell. This can be done by computing $O(m)$ scalar products. The time for the perturbation is dominated by $\mathrm{lp}(m, d)$.

Indeed, to output s^m we solve (at worst) the LP in step {5} with (s_1^m), $(s_1^m, s_2^m)^T$, ..., $(s_1^m, \ldots, s_m^m)^T$.

Similar arguments are used in the proof of (b). The input variables for the recursive calls can be treated as global variables since the calls are tail-recursive. At most one linear program must be handled in memory in one moment. For perturbation,

$\mathcal{O}(size(G))$ memory is sufficient. One witness point is sufficient to be stored at one moment, since w is not necessary anymore at the moment w' is to be found in step {3}. The size of the witness point is $\mathrm{lp}(m,d)$ in the worst case. Thus the overall space complexity is $\mathcal{O}(\mathrm{lp}(m,d))$. □

COROLLARY 3.3. *Algorithm 1 is a compact output-polynomial algorithm for the cell enumeration problem for arrangements.*

4. Avis-Fukuda's Reverse Search – a rival for IncEnu. In this section, we will describe another algorithm for the cell enumeration problem for arrangements of hyperplanes – *the Reverse Search algorithm* [2, 9, 6].

In fact, reverse search is a general meta-algorithmic frame for solving various enumeration problems on graphs. In this paper we only briefly sketch a description of the algorithm and its variants, without a proof of its correctness. We will especially focus on aspects relevant for comparison of the RS algorithm with IncEnu.

4.1. Reverse Search for cell enumeration. The arrangement $\mathcal{A}_m$ can be viewed as a graph $\mathfrak{G} = (\mathcal{C}_m, \{(C(s), C(t)) : C(s), C(t) \text{ are neighbour cells of } \mathcal{A}_m\})$. Hence, nodes are feasible sign vectors (or equivalently: cells), edges are determined by pairs of neighbour sign vectors (or equivalently: by pairs of cells that share common $(d-1)$-dimensional face).

Reverse Search is based on two subroutines: *AdjacencyOracle* and *ParentSearch*. The AdjacencyOracle gives information which vertices of the input graph are neighbour nodes of the currently examined node, and the ParentSearch procedure returns a unique parent of a given node (except for one special node R called *root*). Furthermore, ParentSearch has the property that for every s^m, repeated calls of $s^m := \mathrm{ParentSearch}(s^m)$ will end in R. In other words, ParentSearch defines a covering of $\mathfrak{G}$ by a tree rooted in R.

The sign vector corresponding to the root node is denoted by r. We can w.l.o.g. assume that $r = (1 \dots 1)^T$.

```
{1} Function RevSearchEnu(s^m):
{2}   output s^m
{3}   for each s in AdjacencyOracle(s^m):
{4}     if ParentSearch(s) == s^m:
{5}       RevSearchEnu(s)
```

Algorithm 2: Reverse Search algorithm

With the above subroutines, Reverse Search can be formulated as recursive procedure, see Algorithm 2. The enumeration process is started by RevSearchEnu(r). Then, if RevSearchEnu(s^m) is called, s^m is output and RevSearchEnu is called for each neighbour s of s^m such that s^m is parent node of s. Hence, from every parent, every child is visited, and since every node has the root node as predecessor, every node is output.

As an example, visualization of a sample run of RevSearchEnu can be found in Figure 2 lefts for the same sample arrangement as in Figure 1. The RevSearchEnu calls are ordered from top to bottom.

As mentioned in § 1, there are two variants of Reverse Search. These variants differ in the idea behind Adjacency Oracle. The m-RS algorithm solves a lower number of larger LPs, while the ℓ-RS algorithm solves a higher number of smaller LPs. This

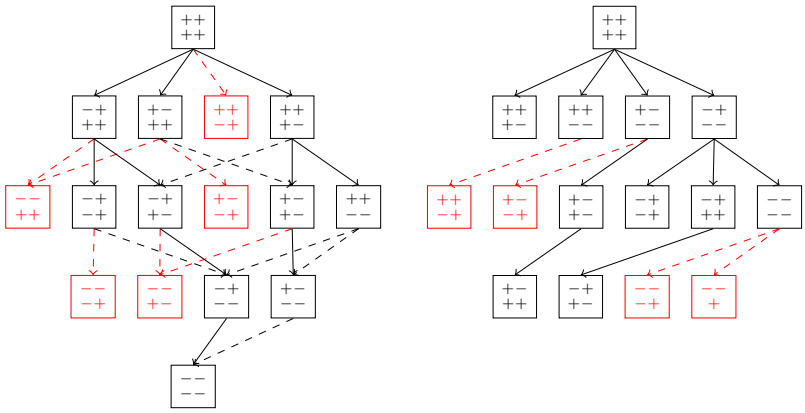


Fig. 2: *Left*: visualisation of RevSearchEnu recursive calls for a sample arrangement with $m = 4$ and $d = 3$. Black nodes: RevSearchEnu calls. Black solid edges: successful flips, ParentSearch passed. Black dashed edge: successful flip, ParentSearch failed. Red nodes: unsuccessful flips.
Right: for comparison, a tree of FlIncEnu for the same arrangement.

difference makes the algorithms directly incomparable. We describe m-RS in § 4.3 and ℓ-RS in § 4.4.

4.2. ParentSearch subroutine. We will assume that each cell $C(s)$ is equipped by a *unique* witness (interior point) $w(s)$, e.g. the one computed by (2.1).

At the beginning of the enumeration process, the witness $w(r)$ is stored. When ParentSearch(s) is called, we find the first hyperplane (denote it by h_j) intersecting $\mathrm{ray}(w(s), w(r))$, then we return $\mathrm{flip}_j(s)$.

The standard lexicographic perturbation can be used if multiple hyperplanes intersect in the same point.

Observe also that each ParentSearch call solves at most one LP (to find the witness point $w(s)$).

Note that the upper bound on recursion depth can be derived from the description of ParentSearch. Since all signs of r are positive, the flip between a cell and its parent cell must be MP-flip. There can be at most m nested MP-flips, hence the maximum recursion depth is also m.

4.3. The *m*-RS algorithm. The original version of RS [2] and also the improved one [6] use the following idea for AdjacencyOracle(s): AdjacencyOracle(s) tries all PM-flips of s and generates the successful ones one by one. Feasibility of a flip can be tested by (2.1).

The restriction to PM-flips follows from the realization of ParentSearch – although some MP-flips of s could be successful, too, they simply cannot pass the ParentSearch test on step {4}.

THEOREM 4.1 ([6]). *The Reverse Search algorithm in m-RS version has time- resp. space-complexity*

$$\mathcal{O}(|\mathcal{C}_m|\ m\ \mathrm{lp}(m,d)) \quad \textit{resp.} \quad \mathcal{O}(\mathrm{lp}(m,d)).$$

The symbol "m" in "m-RS" emphasizes the fact that the LPs solved by AdjacencyOracle always work with all m halfspaces.

4.4. The ℓ-RS algorithm. The ℓ-RS variant of Reverse Search tries to reduce the sizes of LPs to be solved in AdjacencyOracle for the price that in general more LPs must be solved in AdjacencyOracle. The idea of the size-reduction is similar to the one in the paper [8].

The procedure of finding all successful PM-flips is described in Algorithm 3.

Working on the sign vector s, two lists are maintained: a list U of indices of possibly successful PM-flips which have not been examined so far; and a list N of indices of hyperplanes which are known to be tight w.r.t. $C(s)$ (or equivalently: flips of their signs are successful). These lists are initialized in steps $\{2\}$ and $\{3\}$.

The algorithm works iteratively. In each iteration, we choose $j \in U$ and test full-dimensionality of the set

$$E(s,N,j) := \{\xi \in \mathbb{R}^d : \ \forall k \in N \ s_k^m g_k^T \xi > 0, \ -g_j^T \xi < 0\} \tag{4.1}$$

using (2.1). If the test fails, then $\mathrm{flip}_j(s)$ is unsuccessful. Otherwise, the LP (2.1) produces a witness point $\omega \in (E(s,N,j))$. The point ω is separated from witness w of $C(s)$ by h_j (and possibly also by other hyperplanes).

The following easy observation is utilized in step $\{8\}$ to identify one successful flip based on ω:

OBSERVATION 4.2. *Let w be a witness of a cell $C(s)$ and let $a \notin C(s)$. Let h_j be the first hyperplane intersected by the ray(w,a) and assume that it is unique. Then h_j is a tight hyperplane w.r.t. $C(s)$ and $\mathrm{flip}_j(s)$ is successful.*

```
{1}  Function successfulPM-Flips(s, w):
{2}    U := {j : s_j = 1}
{3}    N := ∅
{4}    while U ≠ ∅:
{5}      select arbitrary j from U
{6}      find interior point ω of E(s, N, j) using (2.1)
{7}      if ω:
{8}        k := index of the first hyperplane crossed by ray(w, ω)
{9}        N := N ∪ k; U := U \ k;
{10}     else: U := U \ j
{11}   return N
```

Algorithm 3: Finding successful PM-flips in ℓ-RS

Figure 3: an example of iteration of successful PM-Flips. The idea of hyperplane-searching is depicted in Figure 3. We have an arrangement of four hyperplanes. The $\pm$ signs denote the halfspaces $h_j^{\pm}$. The task is to identify neighbour hyperplanes of $s = (1,1,1,-1)^T$.

Assume we have already found out that $h_1, h_2 \in N$. So $U = \{3\}$ since $s_4^m = -1$. Thus we select $j = 3$ in $\{5\}$ and find an interior point ω of $E(s,N,j)$. In $\{8\}$, the first hyperplane intersected by ray(w,ω) is h_k with $k = 4$. Note that h_3 is a tight hyperplane w.r.t. $C(s)$, too. It will be found in the next (and last) iteration of the cycle, using a new interior point ω'.

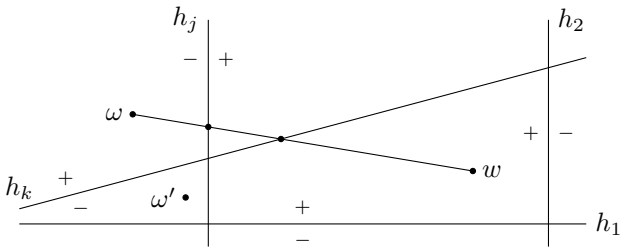


Fig. 3: Idea of identifying tight hyperplanes.

Complexity of AdjacencyOracle using successful PM-Flips. One iteration of successful PM-Flips results in a removal of an index from U or in an addition of an index into N (or both). The number of iterations is at most m. In each iteration, one LP is solved with complexity $\mathrm{lp}(\ell, d)$, where ℓ is the maximum number of tight hyperplanes among all cells of $\mathcal{A}_m$.

THEOREM 4.3 ([9]). *The ℓ-RS algorithm has time complexity $\mathcal{O}(|\mathcal{C}_m|\ (d\ \mathrm{lp}(m,d) + m\ \mathrm{lp}(\ell,d)))$ and space complexity $\mathcal{O}(m^2 + \mathrm{lp}(m,d))$.*

4.5. Brief comparison of m-RS and ℓ-RS. The mechanism of building LPs ensures that no redundant constraints will appear in the LPs to be solved by ℓ-RS. In fact, the sizes of the LPs can be at most $\ell \times d$. However, this advantage of ℓ-RS compared to m-RS is for the price of

(i) maintaining both lists U and N through all nested recursion calls. The size of U plus size of N is at most m, and thus the overall space is $\mathcal{O}(m^2)$;

(ii) the fact that AdjacencyOracle in ℓ-RS has to solve more LPs than m-RS in general. Each iteration of the **while** cycle {4} amounts to one LP to be solved. The number of iterations can be higher than the initial size of the set U, while m-RS solves exactly $|U|$ LPs;

(iii) additional LPs to be solved in ParentSearch calls. On the contrary, m-RS takes the advantage of reusing interior points computed by AdjacencyOracle.

5. The m-IE version. In this Section, we modify FlIE to be directly comparable with m-RS. We start with several notes regarding the differences and similarities between IncEnu and m-RS.

The structure of the algorithm FlIE is quite similar to the Reverse Search scheme in Algorithm 2. The steps {f4} to {f6} are a kind of adjacency oracle. In m-RS, flips to be examined are "filtered" – only PM-flips are examined. In FlIE(s^i, w), only flips of signs with index greater than i are examined, which is actually also a flip-filter. However, we will see in § 5.4 that this filter is more efficient. In particular, note FlIE needs no ParentSearch procedure – the uniqueness of every examined sign vector is ensured by design of the algorithm.

5.1. Sizes of LPs. Note that IncEnu solves LPs of different (smaller) sizes than $m \times d$. More precisely, both algorithms solve at most m linear programs per cell with complexity at most $\mathrm{lp}(m, d)$. In m-RS the LPs are full-sized, while in IncEnu the sizes of LPs vary from $\mathrm{lp}(1, d)$ to $\mathrm{lp}(m, d)$. However, it will (surprisingly) turn out that for the proof of dominance of m-IE it is sufficient to compare the number of LPs to be solved *regardless of their sizes*. Notwithstanding, a smoother analysis taking into

account what exactly happens inside the lp(·) factor (which is treated as a black box here) would be desirable.

5.2. New cell ⇒ some additional LPs. Observe the following behaviour of the algorithms: if a new cell is found during their course, some additional LPs are to be solved. This will be formalized in Proposition 5.2. First we need some notation.

DEFINITION 5.1. *Let all factors influencing the m-RS and FlIE enumeration processes be fixed: the ordering of the generators, the choice of the root cell and the mapping w_u for obtaining unique interior points. For a sign vector s^m, we define*

(a) RS-level *of s^m, denoted $r_{\text{RS}}(s^m)$, is the number of "−" signs in s^m;*

(b) IE-level *of s^m, denoted $r_{\text{IE}}(s^m)$, is the number of fixed signs in the FlIE call at the moment s^m is output.*

IE-level is influenced by the algorithm used for LP and by ordering of generators. So, IE-level must be defined relatively to these factors. A similar fact holds for $r_{\text{RS}}(\cdot)$, too.

PROPOSITION 5.2.

(i) *During the m-RS course, the total number of LPs solved equals to*

$$m|\,\mathcal{C}_m| - \sum\nolimits_{s^m : C(s^m) \in \mathcal{C}_m} r_{\text{RS}}(s^m). \tag{5.1}$$

(ii) *During the IncEnu course, the total number of LPs solved equals to*

$$m|\,\mathcal{C}_m| - \sum\nolimits_{s^m : C(s^m) \in \mathcal{C}_m} r_{\text{IE}}(s^m). \tag{5.2}$$

Proof.

(i) RevSearchEnu in m-RS variant is run for every cell. Consider a cell $C(s^m)$. During AdjacencyOracle, every "+" sign must be checked for flip-feasibility. The number of such signs is $(m - r_{\text{RS}}(s^m))$. No LPs are necessary in ParentSearch.

(ii) FlIE is run for every cell. Consider a call FlIE(s^i, w). In FlIE, the sign vector s^m for the witness w is constructed. A $\text{flip}_j(s^j)$ is checked for feasibility for every predecessor $s^{i+1}, \ldots, s^{m-1}$ of s^m, as well as the $\text{flip}_m(s^m)$. Hence, there are $(m - r_{\text{IE}}(s^m))$ LPs to be solved for s^m. □

5.3. Successful and unsuccessful LPs. Some of the LPs to be solved during the cell enumeration process do find a new witness point (and lead to finding new cells, if the parent search is successful), and some do not. We call these LPs successful resp. unsuccessful.

Lemma 5.3 gives a comparison of the numbers of successful LPs in IncEnu and Reverse Search algorithms. Currently we are aware of no similar straightforward comparison of the numbers of unsuccessful LPs. We leave this as a tempting question.

To claim that IncEnu performs better than m-RS, one should prove that the situation cannot occur that the average RS-level is lower than the average IE-level. Actually, the modification of FlIE to m-IE is necessary just to facilitate the comparison of the numbers of unsuccessful LPs. The main problem is that Reverse Search tests only PM-flips, while IncEnu sometimes must solve some LPs corresponding to MP-flips, too. This obstacle will have to be overcome in the proof of dominance of m-IE.

LEMMA 5.3. *Let f_{IE} and f_{RS} be the number of successful LPs solved in total by FlIE and m-RS, respectively. Then*

$$d\, f_{\text{IE}} \le 2 f_{\text{RS}}. \tag{5.3}$$

Proof. First consider FlIE. LPs are only solved at step {f6}. Every successful LP yields a witness point, say w, and results in outputting a new cell. Thus $f_{\text{IE}} \leq |\mathcal{C}_m|$ holds.

Now consider m-RS. Every successful LP corresponds to one successful flip. Every successful flip corresponds to one facet ($(d-1)$-dimensional cell) of $\mathcal{A}_m$. Conversely, every facet of $\mathcal{A}_m$ is on the boundary of two distinct neighbour cells with sign vectors s, s'. Assume that this facet is in the hyperplane h_j. Then the flip of j-th sign of s or s' is successful. Either s_j or s'_j must be "+", say it is s_j. Hence, RevSearchEnu(s) must solve the LP testing whether the flip of the j-th sign of s is successful. So, for every facet of $\mathcal{A}_m$, one successful LP is solved, and every successful LP solved corresponds to one facet. The number f_{RS} equals to number of facets of $\mathcal{A}_m$.

Since the rank of G is d, every cell has at least d neighbours, which provides the lower bound for the number of facets f_{d-1}. Thus we have $2f_{d-1} \geq d\,|\mathcal{C}_m|$.

Putting it all together we get

$$d\,f_{\text{IE}} \leq d\,|\mathcal{C}_m| \leq 2f_{d-1} = 2f_{\text{RS}}. \qquad \square$$

5.4. Formulation of m-IE – dealing with unsuccessful LPs. Here, we modify FlIE in such a way that it shall solve at most as many unsuccessful LPs as m-RS. We call the resulting algorithm m-IE.

The idea is to make m-IE test only PM-flips, exactly as m-RS does. In FlIE(s^i, w), flips of signs $i+1, \ldots, m$ are tested. Hence, it would be sufficient if we could satisfy the following in m-IE:

(5.4) when m-IE(i) is called, then

all signs of fullsized(w) on positions with indices $\geq i+1$ are 1,

where w is the last found witness point (using FlIE, the call would be FlIE(s^i, w)).

The m-IE in Algorithm 4 is designed to satisfy the property (5.4) while keeping the properties of FlIE. The key is in changing the ordering of generators. Let us describe how it works.

We w.l.o.g assume that m-IE algorithm starts in the cell with sign vector $r = (1, \ldots, 1)^T$. Two global variables are initialized: $w = w(r)$ as an interior point of $C(r)$, and a sign vector $s = s^0$. The property (5.4) is satisfied for m-IE(0) if initialized as above.

Now, assume that we run m-IE(i) such that (5.4) is satisfied. First, the fullsized sign vector s^m corresponding to w is found and output. Then, during the **for** cycle {3} of m-IE, some sucessors of s^i are tested for $j = i+1, \ldots, m$. For each j, we test the sucessor $s^j = \binom{s^{j-1}}{-1}$. The first $j-1$ signs of s^j are the same as s^i has (in particular s^{j-1} is the predecessor of s^m). If s^j generates a cell of $\mathcal{A}_j$, we obtain an interior point ω of $C(s^j)$; this interior point allows us to obtain the sign vector $\sigma :=$ fullSized(ω).

Define the distance between sign vectors s^m, σ as $\text{dist}(s^m, \sigma) := |\{k : s^m_k \neq \sigma_k\}|$, i.e. as the number of different signs between s^m and σ. Note that all indices k with $s^m_k \neq \sigma_k$ must satisfy $k \geq j$, and furthermore we know that $s^m_j \neq \sigma_j$.

If $\text{dist}(s^m, \sigma) = 1$, σ differs from s^m exactly in the j-th sign and no other "$-$" signs are behind the j-th sign. So let us turn to the case $\text{dist}(s^m, \sigma) \geq 2$. Let k and l be the indices of the first two hyperplanes intersected by ray(w, ω); in case of ambiguity, we use the symbolic perturbation. Let α_k and α_l denote the intersections of ray(w, ω)

```
{1}  Function m-IE(i):
{2}    output fullSized(w)
{3}    for each j := i + 1, ..., m:
{4}      s := (s, -1);
{5}      if exists witness ω of C(s):
{6}        k, l, α_k, α_l := findNearestHyperplanes(w, ω)
{7}        w := (α_k + α_l)/2; swap(g_k, g_j)
{8}        m-IE(j)
{9}        s := flip_j(shorten(s, j))
{10}       w := witness point of C(shorten(s, i))
{11}     else:
{12}       s := flip_j(s)
```

Algorithm 4: m-IE algorithm. Global variables w and s are required; initially, w must be interior point of $\mathit{1}^m$.

with h_k and h_l, respectively (see {6}; the procedure findNearestHyperplanes returns the hyperplanes' indices and the intersections).

Then we interchange g_j and g_k and set $\omega := \frac{\alpha_k+\alpha_l}{2}$. Now we have (with one LP) an interior point ω with the property that it is a witness point of $\mathrm{flip}_j(s^m)$, and furthermore, ω corresponds to a full-sized sign vector σ, for which $\mathrm{dist}(s^m, \sigma) = 1$ (step {7}) – this is due to the fact the ray (w, ω) intersects exactly one hyperplane, namely the hyperplane h_j.

The above description leads to the following (informal) corollary:

Corollary 5.4. *Performing generator interchanges from the very beginning of the algorithm course ensures that (5.4) holds for every m-IE call.*

Problems of generators swapping. The interchanges of generators bring some complications:

(i) the generator list changes during the course of the algorithm; one must track these changes and ensure the correctness of the output, and

(ii) to find a generator to swap, it is necessary to know two interior points – one from the current cell, one from the new cell. The interior point from one individual cell must be known at different moments of the course of the algorithm. So there is a question whether and how to store it.

There is a natural solution of (i), similar to the idea of AdjacencyOracle of ℓ-RS: we maintain two lists of generators – a list of used generators and a list of unused generators. In every m-IE call, we move the chosen generator from the list of unused generators to the list of used generators, and when we return back from the recursion, we move back the last generator in the list of used generators. If necessary, each generator can hold its initial ID, and in the output phase, we can sort generators to the initial order in $\mathcal{O}(m)$ time. Though we might need some more memory, it clearly does not affect the order of space complexity.

There are (at least) two solutions of (ii):

- increase space complexity: an interior point can be stored in each level of recursion. Then the space complexity rises to $\mathcal{O}(m \cdot \mathrm{lp}(G))$ (recall that IncEnu and m-RS have the space complexity $\mathcal{O}(\mathrm{lp}(G))$); and
- increase time complexity: an interior point is computed whenever necessary, i.e. an interior point must be computed whenever a new generator swap is to be processed. Every generator swap results in a new cell being output. Hence

there is at most $\mathcal{O}(|\mathcal{C}|)$ additional LPs in total.

From the complexity-theoretic point of view, the latter approach is more efficient. We have employed it in {10} of Algorithm 4. This explains the role of {10} and completes the description of the algorithm.

5.5. Overall properties of m-IE. THEOREM 5.5. *The m-IE algorithm has time complexity*

$$\mathcal{O}(|\mathcal{C}_m|\ m\ \ \mathrm{lp}(m,d))),$$

and space complexity

$$\mathcal{O}(\mathrm{lp}(m,d)).$$

THEOREM 5.6. *Assume the sign vector $s^m = (1,\ldots,1)^T$ generate the cell of $\mathcal{A}_m$. Let λ_{IE} and λ_{RS} stand for the numbers of LPs solved during the course of m-IE and m-RS started from s^m. Then*

$$\lambda_{\text{IE}} \leq \lambda_{\text{RS}} - |\mathcal{C}_m| \left(\tfrac{d}{2} - 2\right). \tag{5.5}$$

Proof. In § 5.3, the distinction between successful and unsuccessful LPs is suggested. Lemma 5.3 proves that the number of successful LPs solved by IE (and FlIE) is at least $d/2$ times better compared to the number of unsuccessful LPs for m-RS. For m-IE, additional "successful" LPs must be solved after the return from some recursive calls in {10}. The number of m-IE calls is $|\mathcal{C}_m|$, so the number of additional successful LPs is also at most $|\mathcal{C}_m|$.

Every unsuccessful LP solved during the course of m-IE amounts to testing a PM-flip from a full-dimensional cell. Since m-RS tests all possible PM-flips, the flips that m-IE tests must be contained among them. The inequality (5.5) follows straightforwardly. □

6. The ℓ-IE version. The modification of IncEnu to be comparable with ℓ-RS has similar background as the construction of m-IE from § 5. The advantage of ℓ-IE is that we will solve smaller LPs. The basic idea is in a combination of m-IE and AdjacencyOracle from ℓ-RS.

6.1. ℓ-RS and m-IE: similarities and differences. Recall how ℓ-RS manages to reduce the sizes of LPs: in each ℓ-RS call, the adjacency oracle builds the set of neighbour hyperplanes which are non-redundant with respect to the current cell. Since the maximal number ℓ of non-redundant hyperplanes among the cells of an arrangement may be significantly lower than m, this approach may bring a complexity advantage against m-RS, despite the fact that ℓ-AO solves in general more LPs than m-AO, and also the fact that ℓ-RS requires additional (full-sized) LPs to treat the ParentSearch procedure. Details are to be found in § 4.4.

Note that the consecutive building of the set of neighbour hyperplanes in AdjacencyOracle of ℓ-RS is a process which has the same structure as m-IE recursive calls: it adds the halfspaces one by one and solves LPs growing in size. When an LP is successful after adding a halfspace h_j^-, this does not necessarily imply that h_j will be used for the next flip – in most cases another (nearest) hyperplane (generator) is found – and this procedure of finding the nearest generator is the same for both ℓ-AO and the m-IE.

The difference is that m-IE(i) always includes the generators $g_1, \ldots, g_i$ to the LPs, unlike ℓ-RS(s^m), which (in the worst case) computes the neighbourhood of these generators from scratch and includes them only if necessary. Recall that it is not clear which approach is better – this is also the reason why we cannot compare m-IE with ℓ-RS (or even with ℓ-IE) directly.

6.2. Employment of AdjacencyOracle of ℓ-RS in m-IE $\Rightarrow$ ℓ-IE. Hence, to make m-IE(i) comparable with ℓ-RS, we try to test the flips of interesting signs (behind the i-th sign), and add the hyperplanes $1, \ldots, i$ to the LPs if necessary – exactly as ℓ-RS does. The resulting ℓ-IE algorithm is summarized as Algorithm 5.

Every ℓ-IE call works with two lists of indices (this is very similar to Adjacency-Oracle of ℓ-RS):

- the list U of unchecked generators. In this list we put all $(m - i)$ free hyperplanes (the hyperplanes corresponding to the last $(m - i)$ signs of s^m), and
- the list N of generators, corresponding to the neighbour hyperplanes found so far.

Note that in ℓ-IE, we no longer use any global variable for a sign vector. For the sake of simplicity, we use one sign vector in every level of recursion. This is possible due to an increased space complexity – now we need space $\mathcal{O}(m^2)$ to store the lists U and N in every level of recursion. The global variable w for a witness point is used and initialized in the same way as in case of m-IE.

The ℓ-IE iterates the **while** cycle {4} until all elements of U are examined.

In each iteration, ℓ-IE takes the first index j from U. For this index we build a cell

$$E(N, s^m, j) := \{\xi \in \mathbb{R}^d : \ \forall i \in N \ s_i^m g_i^T \xi \geq 0, \ g_j^T \xi \leq 0\}$$

and in {6} we check whether it passes test (2.1), i.e. whether there is a witness point ω for it. Note that we take halfspaces induced by neighbour hyperplanes found so far and add the halfspace h_j^- to them. Now ω may be separated from w by several hyperplanes. In step {7} we find the nearest hyperplane and consider it as the actual hyperplane that will be added to N. Note that the procedure findNearestHyperplanes will also return the index l of the second nearest hyperplane for the purposes of computation of the new interior point w in {8}.

If the index k of the nearest hyperplane is in U, then the flip of g_k is successful and we have found a new cell, for which ℓ-IE must be run (step {13}). After the return from the recursion, we must reset the interior point w in the same way as m-IE does (step {14}).

6.3. Overall properties of ℓ-IE. The complexity of ℓ-IE has the same relationship to ℓ-RS as m-IE to m-RS. Its upper bound of both the space complexity and time complexity is the same, and we can only state that ℓ-IE is better by the additive expression $|\mathcal{C}_m| \, (\frac{d}{2} - 2) \, \mathrm{lp}(G)$.

THEOREM 6.1.

(a) The time complexity of ℓ-IE is

$$\mathcal{O}(|\mathcal{C}_m| \ (\mathrm{lp}(m, d) + m \ \ \mathrm{lp}(\ell, d))),$$

and the space complexity is

$$\mathcal{O}(m^2 + \mathrm{lp}(m, d)).$$

(b) *Let w be a witness point for $C(r)$, where $r = (1, \ldots, 1)^T$; r w.l.o.g. generates a cell. Denote λ^m_{IE} and λ^m_{RS} the numbers of $m \times d$ LPs solved during the course of ℓ-IE and ℓ-RS started with interior point w and sign vector r. Then*

$$\lambda^m_{\text{IE}} \leq \lambda^m_{\text{RS}} - |\mathcal{C}_m| \left(\tfrac{d}{2} - 2\right). \tag{6.1}$$

Furthermore, let λ^ℓ_{IE} and λ^ℓ_{RS} denote the numbers of LPs of size at most $\ell \times d$ solved during the course of ℓ-IE and ℓ-RS, respectively, started with the sign vector r with witness point w. Then

$$\lambda^\ell_{\text{IE}} \leq \lambda^\ell_{\text{RS}}. \tag{6.2}$$

Proof. Correctness of (a) follows directly from the construction of the algorithm.

Inequality (6.2) is proved by similar arguments as in Theorem 5.6. Inequality (6.1) follows from the fact that ℓ-RS solves $m \times d$ LPs in ParentSearch subprocedure. The number of ParentSearch calls is at least $\frac{d}{2}|\mathcal{C}_m|$, since each cell has at least d neighbours. □

```
{1}  Function ℓ-IE(i, s^m):
{2}    N := ∅; U := {i + 1, ..., m}
{3}    output s^m;
{4}    while U ≠ ∅:
{5}      select j from U
{6}      if exists witness ω of E(N, s^m, j):
{7}        k, l, α_k, α_l := findNearestHyperplanes(w, ω)
{8}        ω = (α_k + α_l)/2; swap(k, j)
{9}        N := N ∪ j;
{10}       if j ∈ U:
{11}         U := U \ j
{12}         w := ω
{13}         ℓ-IE(j, flip_j(s^m))
{14}         w := witness point of C(s^m)
{15}     else:
{16}       U := U \ j
```

Algorithm 5: ℓ-IE algorithm. A global variable w is used, initially, w is interior point of $(1, \ldots, 1)^T$.

7. Computational experiments. In this section, we provide some initial empirical comparison of the presented algorithms. We premise that the experiments provided evidence for the following conjecture:

CONJECTURE 7.1. *For simple arrangements, IE is $\mathcal{O}(d)$ times faster than m-RS.*

Except for the evidence for the conjecture, we also tried to answer the following general questions:

(Q1) How much better are incremental enumeration algorithms than reverse search algorithms?
We reflect the different approach to the construction of LPs among the algorithms – we compare m- and ℓ-versions of both the algorithms separately. For the sake of completeness of the comparison, besides the m- and ℓ-modifications of IncEnu we also include the original version of IncEnu presented as Algorithm 1) (Notably, it will turn out that the unmodified version is a "winner".)

(Q2) Is the ℓ-version or the m-version of RS better? And what about the m- and ℓ-versions of IE?

7.1. Design of the experiments.

Comparison criteria. All the algorithms strongly rely on linear programming – linear programming is surely the most time-consuming (and actually also the most often called) subprocedure. We have also taken the numbers of LPs as the comparison criterion in the theoretical comparison in § 5 and § 6.

For (Q1), we take the same approach – the main criterion for comparison is the number of LPs. The sizes of LPs in reverse search algorithms are not smaller than the sizes of LPs in incremental enumeration algorithms. We also present elapsed times.

For the question (Q2), the sizes of the LPs are crucial, because the worst-case complexity of the algorithms depends on the sizes. Hence, the comparison criterion for the question (Q2) will be elapsed time.

Implementation. We implemented the presented algorithms in Python 2.7. We used the `gurobi` solver ([7]) to provide us the capability for fast isCell and ParentSearch procedures.

7.2. Test cases. We compared the algorithms for representatives of several classes of arrangements. The classes differ in *degeneracy* of underlying arrangements.

Nondegenerate (simple) and degenerate arrangements. An arrangement spanned by $g_1, \dots, g_n \in \mathbb{R}^d$ is *nondegenerate* (*simple*) if no $(d-1)$-tuple of vectors $g_1, \dots, g_n$ is linearly dependent; otherwise it is *degenerate*. Said otherwise, the arrangement is nondegenerate if $g_1, \dots, g_n$ are in a general position. A non-degenerate arrangement has the maximal possible number of faces.

A brief description of the test classes of arrangements will follow.

Randomly generated arrangements. Non-degenerate random arrangements of different sizes were generated for $2 \le d \le 7$ and $10 \le m \le 50$. The generators had integer entries between -50 and 50. (To be more precise, non-degeneracy of the arrangements was tested ex post using the number of cells enumerated. We used the fact that for non-degenerate arrangements, upper bound of number of cells is attained.)

Degenerate random arrangements were generated in a similar way. The amount of degeneracy was controlled with *degeneracy ratio* r. First, d random generators were generated. Second, each following generator g_i $(i > d)$ was chosen as follows: with probability r, g_i was chosen as a linear combination of a random subset of generators $g_1, \dots, g_{i-1}$; with probability $1-r$, g_i was chosen at random.

In the table of results, these instances are denoted by triple (m, d, r).

Arrangements combinatorially dual to permutahedra. Permutahedra are simple zonotopes. Arrangements corresponding to permutahedra are highly degenerated and have the property that every cell has d neighbour cells, i.e. as few as possible.

We tested affine arrangements corresponding to permutahedra in dimensions 5 to 9. An affine arrangement corresponding to the permutahedron in $\mathbb{R}^d$ is denoted by *perm d*.

The asymptotically worst arrangements. Let e_i denote the i-th unit vector in $\mathbb{R}^d$ and consider normals of hyperplanes determining the arrangement $\mathcal{A}_m$ in the following form:

$$g_i := \begin{cases} e_i & \text{for } i = 1, \dots, d, \\ c_{i1} g_{d-1} + c_{i2} g_d & \text{for } i = d+1, \dots, m, \end{cases} \tag{7.1}$$

where c_{i1} and c_{i2} $(d < i \le m)$ are nonzero coefficients such that no two normals are linearly dependent. It can be proved that these arrangements are the asymptotically

worst possible for (Fl)IE and m-RS, since the number of LPs to be solved per cell is $\Omega(m/2)$. See Appendix A for proof.

Since these arrangements have one complex $(d-2)$-dimensional face, usually called *ridge*, we denote it by $r\ m, d$.

7.3. Results. The number of LPs (or #LPs for short) and elapsed times for individual instances are shown in Table 1. In the last column of the table there is a ratio "#LPs of m-RS/#LPs of IE", which led us to the formulation of Conjecture 7.1.

Several observations about the behaviour of the algorithms can be made:

- The m-versions of the algorithms seem to perform better than ℓ-versions. However, this may be affected by properties of the LP-solver used. `Gurobi` tends to reuse the information from the last optimization and also seems to have an overhead when a constraint is added to a model. So, m-versions of the algorithms have a big advantage, because consecutive LPs differ often only in one constraint, unlike ℓ-versions, where LPs are often rebuilt from scratch.
  If we modify the implementation in a way that no information about the LP previously solved is reused by the solver, ℓ-RS becomes better than m-RS, ℓ-IE becomes better than m-IE and for large m even better than IE.
  We consider the necessity of rebuilding the LPs as the property of the algorithm rather than the implementation issue. Hence, the answer to (Q2) is that m-versions of algorithms are faster.
- We proved in § 5.4 (for m-IE) and § 6 (for ℓ-IE) that incremental enumeration algorithms perform better than reverse search algorithms. Additionally, in non-degenerate or slightly degenerate cases, incremental enumeration algorithms seem to be $\mathcal{O}(d)$ times better. This is what we have stated in Conjecture 7.1 for IE, which has the least #LPs among the incremental enumeration algorithms.
- Although IE has the best overall #LPs, we cannot conjecture that it is better than m-IE, since m-IE can attain lower number of unsuccessful LPs in general.

8. Conclusions. We presented a new incremental algorithm IE for enumeration of full-dimensional cells of hyperplane arrangements. IE is compact and output-sensitive, with worst-case time complexity of $\mathcal{O}(|\mathcal{C}_m|\, m\ \mathrm{lp}(G))$ and space complexity $\mathcal{O}(\mathrm{lp}(G))$.

We built two versions of our algorithm (m-IE and ℓ-IE) to allow for theoretical comparison with two versions of the Reverse Search algorithm (m-RS and ℓ-RS) by Avis and Fukuda [2, 9, 6]. While our algorithms have the same worst-case complexity bounds (Theorems 5.5 and 6.1(a)), they are faster than variants of Reverse Search. In fact, we showed that regardless of which version of Reverse Search one chooses, our algorithm performs better (Theorems 5.6 and 6.1(b)).

We have also implemented the algorithms and made some initial computational experiments. For simple arrangements, our IE algorithm seems to have time complexity $\mathcal{O}(d)$ times better than m-RS algorithm (we stated this interesting observation in Conjecture 7.1; this conjecture should be decided in future work). Similarly promising results were achieved for slightly degenerate arrangements and for the other versions of the algorithms.

[1] Kim Allemand, Komei Fukuda, Thomas M. Liebling, and Erich Steiner, *A polynomial case of unconstrained zero-one quadratic optimization*, Mathematical Programming, 91 (2001), pp. 49–52.
[2] David Avis and Komei Fukuda, *Reverse search for enumeration*, Discrete Appl. Math., 65 (1996), pp. 21–46.
[3] Robert Creighton Buck, *Partition of Space*, The American Mathematical Monthly, 50 (1943), pp. 541–544.
[4] Michael R. Bussieck and Marco E. Lübbecke, *The vertex set of a 01-polytope is strongly P-enumerable*, Computational Geometry, 11 (1998), pp. 103 – 109.
[5] Herbert Edelsbrunner, Joseph O'Rouke, and Raimund Seidel, *Constructing arrangements of lines and hyperplanes with applications*, SIAM J. Comput., 15 (1986), pp. 341–363.
[6] Jean Albert Ferrez, Komei Fukuda, and Thomas M. Liebling, *Solving the fixed rank convex quadratic maximization in binary variables by a parallel zonotope construction algorithm*, European Journal of Operational Research, 166 (2005), pp. 35 – 50.
[7] Gurobi Optimization, Inc., *Gurobi Optimizer Reference Manual*, 2014.
[8] Th Ottmann, S. Schuierer, and S. Soundaralakshmi, *Enumerating Extreme Points in Higher Dimensions*, 1995.
[9] Nora Sleumer, *Output-sensitive cell enumeration in hyperplane arrangements*, in Algorithm Theory — SWAT'98, Springer Berlin Heidelberg, 1998, pp. 300–309.
[10] Thomas Zaslavsky, *Facing up to arrangements: Face-count formulas for partitions of space by hyperplanes*, no. 154 in Memoirs of the American Mathematical Society, American Mathematical Society, 1975.

Appendix A. Tightness of the worst-case time complexity bound for (7.1)**.** In this section we show that the worst-case time complexity bound of IncEnu algorithms cannot be improved. In particular, we prove that the number of LPs to be solved by IncEnu is $\Omega(m \cdot |\mathcal{C}_m|)$ for the arrangements in form (7.1). Then it is not difficult to conclude that overall complexity of IncEnu is $\Omega(|\mathcal{C}_m|\, m\, \mathrm{lp}(G))$.

Observation A.1. *Assuming $i \geq d$, the number of cells of the arrangement $\mathcal{A}_i$ with generators* (7.1) *is $|\mathcal{C}_i| = 2^{d-1}(i-d+2)$.*

Proof. First, $\mathcal{A}_{d-2}$ has 2^{d-2} cells. Second, all remaining generators, $g_{d-1}, \ldots, g_i$ (denote the set of them by G'_i) are orthogonal to $g_1, \ldots, g_{d-2}$, and hence their hyperplanes will dissect every cell of $\mathcal{A}_{d-2}$. The hyperplanes from G'_i form an arrangement $\mathcal{A}'_i$. Since $\mathrm{rank}(G'_i) = 2$, the arrangement is 2-dimensional, hence it is nondegenerate and it has $2(m-d+2)$ cells due to (1.1). □

Lemma A.2. *The number of LPs solved by IncEnu during cell enumeration of $\mathcal{A}_m$ with generators* (7.1) *is*

$$m\, 2^{d-1}(m-d+2) - \left(\sum_{i=0}^{d-2} 2^i i + \sum_{i=d-1}^{m} 2^{d-1} i \right). \tag{A.1}$$

Proof. According to Observation 5.2, the number of LPs solved by IncEnu equals to $m\,|\mathcal{C}_m|$ reduced by the sum of IE-levels of all cells, which is exactly the formula (A.1): let j satisfy $d-1 \leq j \leq m$. At j-th level of IncEnu recursion,

$$|\mathcal{C}_j| - |\mathcal{C}_{j-1}| = 2^{d-1}(j-d+2-(j-1-d+2)) = 2^{d-1}$$

new cells are found in total. Hence, 2^{d-1} cells have IE-level j. Numbers of cells with IE-level k such that $1 < k < d-1$ are easily observed, since every h_k doubles the number of cells of $\mathcal{A}_{k-1}$. □

Corollary A.3. *For every fixed d and $m \geq d-1$, the total number of LPs solved during cell enumeration of $\mathcal{A}_m$ with generators* (7.1) *is $\Omega(|\mathcal{C}_m|\, m)$.*

Proof. The mean number of LPs per cell is

$$\frac{m\,|\mathcal{C}_m| - \left(\sum_{i=0}^{d-2} 2^i i + \sum_{i=d-1}^{m} 2^{d-1} i\right)}{|\mathcal{C}_m|} =$$

$$= m - \frac{2^{d-1}(d-2) - 2^{d-1} + 2 + 2^{d-1}(m+d-1)(m+2-d)/2}{2^{d-1}(m-d+2)} =$$

$$= m - \frac{m+d-1}{2} - \frac{d-3}{m-d+2} - \frac{1}{2^{d-2}(m-d+2)} = I(m).$$

It is easy to see that $I(m) \to \frac{m}{2}$ with $m \to \infty$. This implies that the total number of LPs is $\Omega(|C_m|\, m)$. □

Name	m	d	$\lvert\mathcal{C}_m\rvert$	IE		m-IE		ℓ-IE		m-RS		ℓ-RS		m-RS/IE
10,3,0	10	3	176	386	0.11	625	0.20	886	0.48	938	0.33	1720	0.92	2.43
10,4,0	10	4	386	638	0.19	1083	0.38	1898	1.06	2061	0.76	4327	2.18	3.23
10,5,0	10	5	638	848	0.26	1493	0.61	2846	1.77	3265	1.28	7603	4.17	3.85
10,6,0	10	6	848	968	0.36	1833	0.72	3367	2.22	4243	1.89	10562	6.57	4.38
20,3,0	20	3	1351	6196	1.64	9606	2.67	13019	6.73	13655	4.88	20146	10.10	2.20
20,4,0	20	4	6196	21700	6.53	35094	21.03	63168	47.27	67319	33.67	108172	70.86	3.10
20,5,0	20	5	21700	60460	20.18	102258	40.67	212230	144.59	228029	139.59	403846	321.52	3.77
20,6,0	20	6	60460	137980	47.58	242337	113.51	591343	396.65	612998	330.02	1188794	928.27	4.44
20,7,0	20	7	137980	263950	103.67	486624	297.99	1367735	1452.67	1462135	770.79	3006679	2403.59	5.54
50,2,0	50	2	1275	20858	5.64	26056	6.90	29925	9.52	40767	17.45	45358	32.11	1.95
40,3,0	40	3	10701	102091	46.56	159044	63.02	223070	118.86	244996	114.21	300212	162.69	2.40
35,4,0	35	4	59536	384168	139.18	656659	244.23	1110671	674.81	1142383	591.36	1545110	1064.73	2.97
25,5,0	25	5	68406	245506	99.77	422610	203.64	914183	673.30	924784	500.84	1492409	1189.67	3.77
25,6,0	25	6	245506	726206	268.34	1347559	590.40	3374194	2316.66	3420344	1759.51	5874270	4391.67	4.71
20,2,0.7	20	2	194	1241	0.36	1707	0.46	1969	0.77	2218	0.69	2862	1.16	1.79
20,3,0.7	20	3	1241	5516	1.58	8246	2.70	13374	6.31	14767	5.24	20922	10.25	2.68
20,4,0.7	20	4	5311	19469	6.00	31298	12.90	53729	34.12	60400	29.24	94815	68.19	3.10
20,5,0.7	20	5	19680	54297	22.56	95716	47.69	195542	150.02	203450	96.40	359490	245.96	3.75
20,6,0.7	20	6	50509	121119	45.63	195447	83.68	491491	340.50	514979	283.77	991952	781.52	4.25
20,7,0.7	20	7	129639	253160	94.35	430304	195.31	1240857	865.87	1301493	689.00	2720447	2073.79	5.14
20,2,0.9	17	2	108	653	0.15	735	0.18	753	0.28	866	0.24	1167	0.40	1.33
20,3,0.9	20	3	1036	5031	1.36	7186	2.21	11441	6.04	12829	6.38	17960	10.46	2.55
20,4,0.9	20	4	4850	16124	3.94	25482	7.19	44507	19.64	49805	16.65	80827	49.05	3.09
20,5,0.9	20	5	17933	49832	17.01	82150	25.78	173148	91.84	176403	86.09	315931	225.03	3.54
20,6,0.9	20	6	52858	131642	37.86	207350	79.57	560231	350.52	522767	259.02	1026226	695.72	3.97
20,7,0.9	20	7	108470	212495	77.57	368776	177.08	1170209	906.64	1089151	676.27	2251160	1920.31	5.13
perm 5	9	4	60	157	0.03	222	0.08	251	0.14	311	0.11	514	0.22	1.98
perm 6	14	5	360	1211	0.32	1541	0.50	2263	1.11	2761	0.85	4346	2.40	2.28
perm 7	20	6	2520	10417	2.87	13873	4.64	21294	11.40	26041	7.83	39521	20.69	2.50
perm 8	27	7	20160	99155	21.00	133152	34.71	277692	124.36	292321	83.60	424459	207.55	2.95
perm 9	35	8	181440	1036897	282.56	1321175	492.25	3136350	1846.16	3356641	1176.27	4735670	3305.06	3.24
r 20,2	20	2	40	382	0.09	421	0.09	456	0.16	401	0.11	477	0.13	1.05
r 30,2	30	2	60	872	0.22	931	0.22	926	0.29	901	0.29	1017	0.30	1.03
r 40,2	40	2	80	1562	0.38	1641	0.39	1716	0.55	1601	0.54	1757	0.49	1.02
r 20,4	20	4	144	1232	0.30	1375	0.35	1398	0.49	1441	0.46	1926	0.85	1.17
r 20,5	20	5	272	2192	0.51	2463	0.60	2671	1.13	2721	0.79	3847	1.84	1.24

Table 1: Results of experiments. For every algorithm, left number is #LPs, right number is elapsed time. The last column is ratio of #LPs for m-RS and #LPs for IncEnu.