SIAM J. DISCRETE MATH.
Vol. 32, No. 1, pp. 455–473
© 2018 Society for Industrial and Applied Mathematics

Downloaded 10/20/22 to 128.93.162.241 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

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

MIROSLAV RADA† AND MICHAL ČERNÝ‡

Abstract. We design a new algorithm, called Incremental Enumeration (IncEnu), for the enumeration of full-dimensional cells of hyperplane arrangements (or dually, for the 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 and Fukuda's Reverse Search (RS), including its later improvements by Sleumer and others. The basic versions of IncEnu and RS are not directly comparable, since they solve different numbers of linear programs (LPs) of different sizes. We therefore reformulate our algorithm as a version that permits comparison with RS in terms of the number 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 bounds). In spite of the fact that IncEnu and RS have the same asymptotic bounds, it is proved that IncEnu is faster than RS by a nontrivial additive term. Our computational experiments show that for most test cases IncEnu is significantly faster than RS in practice. Based on the results obtained, we conjecture that IncEnu is $\mathcal{O}(d)$ times faster for nondegenerate arrangements, where d denotes the dimension of the arrangement.

Key words. arrangements, cell enumeration, zonotopes, output-sensitive algorithm, compact algorithm

AMS subject classification. 52C35

DOI. 10.1137/15M1027930

1. Introduction.

We design a new algorithm, called the *Incremental Enumeration Algorithm* (IncEnu, IE), for the enumeration of (full-dimensional) cells of an arrangement of m hyperplanes in $\mathbb{R}^d$, or dually, for the 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 the input ("compactness"). The number of cells in an arrangement can be as large as [3, 5, 10]

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

which is superpolynomial in m in general, for example if $d = m/2$.[1] Thus output-polynomiality is "the best" that 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 and Fukuda's Reverse Search (RS) [2] is considered a standard; RS is an output-polynomial and compact algorithm.

*Received by the editors June 29, 2015; accepted for publication (in revised form) August 25, 2017; published electronically February 13, 2018.
http://www.siam.org/journals/sidma/32-1/M102793.html
Funding: The work was supported by Czech Science Foundation grant P403/16-00408S.
†Faculty of Finance and Accounting, University of Economics, Prague, nám. W. Churchilla 4, 130 67 Prague 3, Czech Republic (miroslav.rada@vse.cz).
‡Faculty of Informatics and Statistics, University of Economics, Prague, nám. W. Churchilla 4, 130 67 Prague 3, Czech Republic (cernym@vse.cz).
[1]The special case $d = \mathcal{O}(1)$, for which $\zeta(m,d)$ is polynomial in m, has important applications in quadratic programming [1, 6].

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

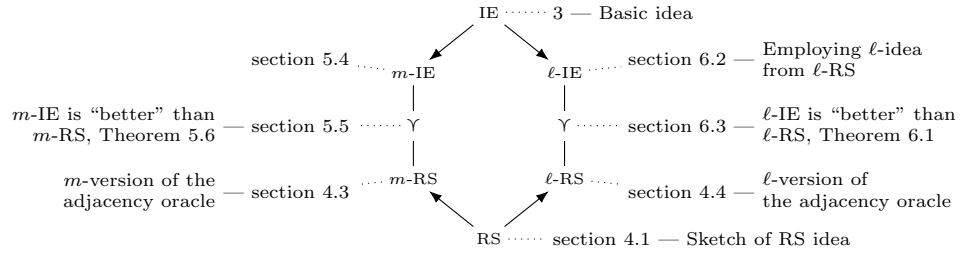
Downloaded 10/20/22 to 128.93.162.241 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

For both approaches, IE and RS, the most time-consuming subprocedure is linear programming. Unfortunately, IE and RS are not directly comparable since they solve different numbers of linear programs (LPs) of different sizes. Most of the work in this paper is devoted to designing a method of comparing IE and RS.

RS is currently known in (at least) two versions, called m-RS [6] and ℓ-RS [9], which differ in the method used for listing neighboring cells of a given cell (known as the *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 uses, the corresponding version of IncEnu performs better in terms of time.*

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

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



In section 7 we present some computational experiments carried out in order to compare IE, m-IE, ℓ-IE, m-RS, and ℓ-RS. The empirical evidence indicates that IE performs the fastest.

2. Preliminaries.

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

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 $h_i^- = \{x \mid g_i^T x \leq 1\}$.

Neighbors and flips. Let s, s' be sign vectors differing in exactly one sign, say the ith. We say that s' results from s by a *plus-to-minus flip*, or *PM-flip* for short, if

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

$s_i = 1$ and $s'_i = -1$. Similarly, s' results from s by a *minus-to-plus flip (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 *neighbors* and the (unique) hyperplane separating them is called the *neighboring hyperplane*.

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

Notation. We denote dimensions by upper indices. In particular, the upper index in $s^i \in \{\pm 1\}^i$ emphasizes that s^i is a sign vector with i signs. We will use the empty sign vector for convenience several times. It is denoted by s^0.

We will often concatenate sign vectors. To avoid confusion with the binomial coefficient, we stress that the symbol $\binom{s}{t}$ denotes concatenation of s and t.

Arrangements $\mathcal{A}_i$ and 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 sign vector $s^i \in \{\pm 1\}^i$ determine cell $C(s^i) \in \mathcal{C}_i$. For $j > i$, the concatenation $s^j = \binom{s^i}{t^{j-i}}$ of s^i and some $t^{j-i} \in \{\pm 1\}^{j-i}$ is called an *extension* of s^i; if $j = i + 1$, it is an *immediate* extension.

An interior point of a cell C is called a *witness point*, or *witness* for short. A witness point for a cell generated by sign vector s^i is denoted by $w(s^i)$. The following useful property can be assumed without loss of generality: if $j > i$, then a witness ξ of $C(s^i) \in \mathcal{C}_i$ is also a witness of $C(s^j) \in \mathcal{C}_j$ for some extension $s^j \in \{\pm 1\}^j$ of s^i. (The degenerate case $g_\iota^T \xi = 1$ for some $\iota \in \{i + 1, \dots, j\}$ can easily be resolved by a perturbation of ξ.)

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

Notation. Let G_i denote the matrix with columns $g_1, \dots, g_i$; we set $G := G_m$. Given a vector z, $\mathrm{diag}(z)$ is the square matrix with the entries of z as its diagonal entries. The symbol $\mathbf{1}$ stands for the vector of ones of appropriate dimension.

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: $C(s^i) \in \mathcal{C}_i$ if and only if

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

Note that this LP also provides through ξ a witness of $C(s^i) \in \mathcal{C}_i$ (if it exists). Using a deterministic LP solver, we can without loss of generality assume that each cell is equipped with a unique interior point. We denote the witness point computed by (2.1) by $w(s)$ for a cell $C(s)$. This will be essential in section 4.2 and later, where the positions of witnesses affect the enumeration process.

Suppose 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*.

The symbol $\mathrm{lp}(\mu, d)$ denotes the space or time complexity needed to solve a linear program with $\mu + \mathcal{O}(1)$ constraints and $d + \mathcal{O}(1)$ variables with bit-size data bounded by the bit-size of G. When possible, we use the symbol $\mathrm{lp}(G)$ as shorthand for $\mathrm{lp}(m, d)$.

3. The basic algorithm: IncEnu. We present the basic algorithm in two equivalent formulations: IE and Flip-based IE (FlIE). The reason for doing so is that IE is conceptually very simple and easily verified to be correct, while FlIE is in a form that is quite similar to the RS algorithm, which allows for an easier comparison of these two algorithms.

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 10/20/22 to 128.93.162.241 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

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

{1} **Function** IE(s^i, w):	{f1} **Function** FlIE(s^i, w):
{2} **if** $i < m$:	{f2} $s^m :=$ fullSized(w)
{3} **if** $w \in h^+_{i+1}$: $\sigma = 1$ **else**: $\sigma = -1$	{f3} **output** s^m
{4} IE$(\binom{s^i}{\sigma}, w)$	{f4} **for** $j = m, m-1, \ldots, i+1$:
{5} **if** there exists witness w' of $C\binom{s^i}{-\sigma}$:	{f5} $s^j :=$ flip$_j$(shorten(s^m, j))
{6} IE$(\binom{s^i}{-\sigma}, w')$	{f6} **if** there exists witness w' of $C(s^j)$:
{7} **else**:	{f7} FlIE(s^j, w')
{8} **output** s^i	

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. This enumerates all the cells of $\mathcal{A}_m$ when run as IncEnu$(s^0, 0)$, where s^0 is the 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 as well.

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

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

(a) the whole cell $C(s^i)$ belongs to one of the halfspaces h^+_{i+1} or h^-_{i+1}, or

(b) the hyperplane h_{i+1} subdivides $C(s^i)$ into two new cells of $\mathcal{A}_{i+1}$.

First, in step {3} the algorithm checks which of the extensions $\binom{s^i}{\pm 1}$ of s^i will surely generate a cell of $\mathcal{A}_{i+1}$. Then IE is run for this extension in step {4}.

Second, the cases (a) and (b) are actually distinguished in step {5} using the test (2.1). In case (b), a new witness point w' is found in {5} and IE is called recursively in {6}.

Solving the LP in {5} is 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. This will be done in Theorem 3.2 in section 3.4.

3.2. FlIE: The version more comparable with RS. Let us first define two helper functions. The function fullSized(w) constructs a sign vector s^m such that w is the witness for $C(s^m)$. Ties can be resolved arbitrarily. The procedure shorten(s^i, j) with $i > j$ cuts off the last $(i - j)$ signs of s^i.

The FlIE algorithm reformulates IE in the following way: the extensions of s^i by σ, performed and processed in step {4} during the recursion, are collapsed into a single call of fullSized(w) in step {f2} of FlIE. The recursive calls of IE in step {4} are not required in FlIE.

The **for** loop at steps {f4}–{f7} is an analogy of {5}–{6} in a sequence of consecutive calls of IE. It takes s^m and shortens it to the appropriate length at each iteration to obtain the s^i corresponding to that obtained in the IE call.

Running the FlIE algorithm can also be viewed as *fixing signs*. Initially, a witness w of a cell $C(s^m)$ is given and no sign in s^m is fixed. In a FlIE(s^i, w) call, the first i signs of s^m are *fixed* and only the signs with indices $i+1, \ldots, m$ are flipped.

3.3. Example run of IE and FlIE. Runs of both IE and FlIE are visualized in Figure 1. Each black edge corresponds to finding one new witness point and a new recursive call of IE (respectively, FlIE). Green (dashed) edges and nodes correspond

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 10/20/22 to 128.93.162.241 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

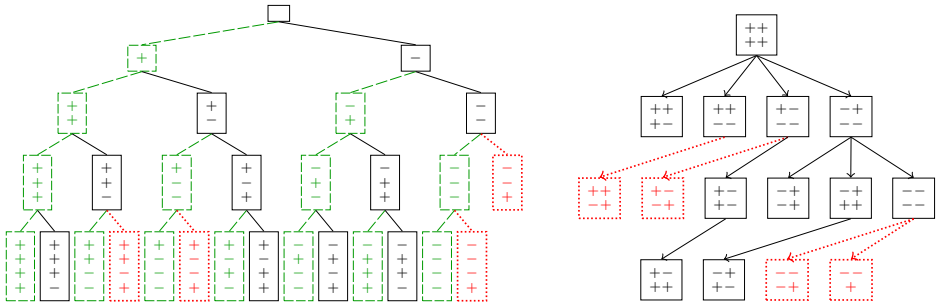


FIG. 1. *IE (left tree) and FlIE (right tree) for a sample arrangement with $m = 4$ and $d = 3$. Red (dotted) nodes: infeasible sign vectors (decided by LP). Green (dashed) nodes: feasible sign vectors found without LP. Black nodes: feasible sign vectors found with LP.*

to "free" extensions without LP. Note that these are "collapsed" in the right picture. Red (dotted) nodes and edges correspond to infeasible LPs.

3.4. Complexity and correctness of IncEnu. We summarize the main properties of IncEnu. It can be shown that the same results hold true for FlIE as well.

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 $\mathcal{A}_m$ that is complete and contains no duplicates.*

Proof.

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

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 the ith level of recursion. For each $C(s^i) \in \mathcal{C}_i$ generated by s^i, both extensions s^{i+1} of s^i are tested to see 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, then we obtain 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 each IE call either a sign vector s^m generating a cell is output or at least one more IE procedure is called. During an IE run, at most one linear program with complexity $\mathrm{lp}(G)$ is solved. To output s^m we solve (at worst) the LP in step {5} with (s_1^m), $(s_1^m, s_2^m)^T$, ..., $(s_1^m, \ldots, s_m^m)^T$.

To prevent the degenerate case $g_{i+1}^T w = 1$ for some i, we also need the 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 cell being examined. This can be done by computing $O(m)$ scalar products. The time for perturbation is dominated by $\mathrm{lp}(m, d)$.

Similar arguments apply to 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 at any given moment. For perturbation, $\mathcal{O}(size(G))$ memory is sufficient. Only one witness point need be stored at any given

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 10/20/22 to 128.93.162.241 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

Downloaded 10/20/22 to 128.93.162.241 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

moment, since w is not required when w' is to be found in step $\{3\}$. The size of the witness point is $\mathrm{lp}(m,d)$ at worst. So, 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 and Fukuda's Reverse Search: A rival for IncEnu. We describe another algorithm for the cell enumeration problem for arrangements of hyperplanes: *the Reverse Search algorithm* [2, 9, 6].

Reverse Search is a framework for solving various enumeration problems on graphs. We just briefly sketch the algorithm and its variants, without proving correctness. We focus especially on those aspects relevant to the 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 $\mathfrak{G} = (\mathcal{C}_m, \{(C(s), C(t)) : C(s), C(t) \text{ are neighboring cells of } \mathcal{A}_m\})$. Hence, nodes are feasible sign vectors (or, equivalently, cells), and edges are determined by pairs of neighboring sign vectors (or, equivalently, by pairs of cells that share a common $(d-1)$-dimensional face).

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

The sign vector corresponding to the root node is denoted by r. We can without loss of generality assume that $r = (1, \ldots, 1)^T$.

With the above subroutines, Reverse Search can be formulated as a 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 neighbor s of s^m such that s^m is the parent of s. From each parent every child is therefore visited, and so every node is output, since every node has the root as a predecessor.

A visualization of a sample run of RevSearchEnu is shown in Figure 2 for the same sample arrangement as in Figure 1. The RevSearchEnu calls are ordered from top to bottom.

As mentioned in section 1, there are two variants of Reverse Search. These variants differ in the idea behind AdjacencyOracle. The m-RS algorithm solves a lower number of larger LPs, while the ℓ-RS algorithm solves a higher number of smaller LPs. This difference makes the algorithms not directly comparable. We describe m-RS in section 4.3 and ℓ-RS in section 4.4.

4.2. ParentSearch subroutine. Note that the ParentSearch subroutine utilizes the assumption from section 2: we are assuming that each cell $C(s)$ is equipped by its *unique* witness (interior point) $w(s)$ computed by (2.1).

Algorithm 2. Reverse Search algorithm.

```
{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)
```

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

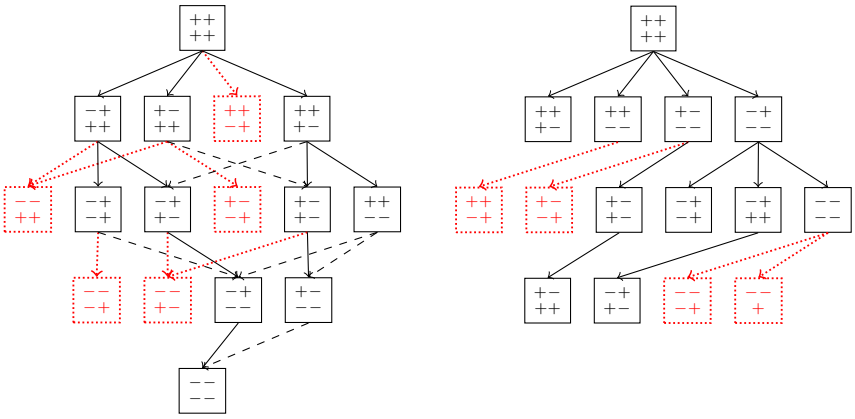


FIG. 2. *Left: visualization of RevSearchEnu recursive calls for a sample arrangement with* $m = 4$ *and* $d = 3$. *Black nodes: RevSearchEnu calls. Black solid edges: successful flips, ParentSearch passed. Black dashed edges: successful flip, ParentSearch failed. Red (dotted) nodes: unsuccessful flips.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

At the beginning of the enumeration process, the witness $w(r)$ is stored. When ParentSearch(s) is called, we find the first hyperplane (denoted by h_j) intersecting $\text{ray}(w(s), w(r))$, then we return $\text{flip}_j(s)$. Standard lexicographic perturbation can be used if multiple hyperplanes intersect at the same point.

Each ParentSearch call solves at most one LP (to find the witness point $w(s)$). An upper bound on recursion depth can be derived from the description of ParentSearch. Since all the signs of r are positive, the flip between a cell and its parent cell must be an MP-flip. There can be at most m nested MP-flips, so that the maximum recursion depth is m.

4.3. The *m*-RS algorithm. Both the original version of RS [2] and 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). Restriction to PM-flips is possible due to the realization of ParentSearch—although some MP-flips of s could also be successful, they simply cannot pass the ParentSearch test in step {4}.

THEOREM 4.1 (See [6]). *The m-RS version of the Reverse Search algorithm has time complexity* $\mathcal{O}(|\mathcal{C}_m|\ m\ \text{lp}(m,d))$ *and space complexity* $\mathcal{O}(\text{lp}(m,d))$.

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 for finding all successful PM-flips is described in Algorithm 3.

Two lists are maintained when working on a sign vector s: a list U of indices of possibly successful PM-flips which have so far not been examined, and a list N of indices of hyperplanes which are known to be neighboring with respect to $C(s)$ (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 the full-dimensionality of the set

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 10/20/22 to 128.93.162.241 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

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

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

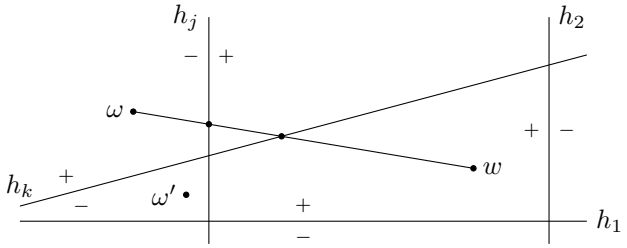


FIG. 3. *Idea behind identifying neighboring hyperplanes.*

$$E(N,s,j) := \left\{\xi \in \mathbb{R}^d :\ \forall k \in N\ s_k\left(g_k^T\xi - 1\right) \geq 0,\ g_j^T\xi \leq 1\right\} \tag{4.1}$$

using (2.1). If the test fails, then $\text{flip}_j(s)$ is unsuccessful. Otherwise the LP (2.1) produces a witness point $\omega \in E(N,s,j)$. 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 {7} to identify one successful flip based on ω.

Observation 4.2. Let w be a witness of cell $C(s)$ and let $a \notin C(s)$. Let h_j be the first hyperplane intersected by $\text{ray}(w,a)$ and assume that it is unique. Then h_j is a neighboring hyperplane with respect to $C(s)$ and $\text{flip}_j(s)$ is successful.

Figure 3*: An example of the iteration of successfulPM-Flips.* The idea behind 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 the neighboring hyperplanes of $s = (1,1,1,-1)^T$.

Assume that 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 $E(N,s,j)$. In {8}, the first hyperplane intersected by $\text{ray}(w,\omega)$ is h_k with $k = 4$. Note that h_3 is a neighboring hyperplane with respect to $C(s)$ as well. It will be found in the next (and last) iteration of the cycle, using a new interior point ω'.

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

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

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 10/20/22 to 128.93.162.241 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

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 at the price of what follows.

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

(ii) AdjacencyOracle in ℓ-RS having to solve more LPs than m-RS in general. Each iteration of the while loop {4}–{9} 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. These LPs are accounted for in the term $O(d\ \mathrm{lp}(m,d))$ in Theorem 4.3. The additional LPs result from the fact that ParentSearch requires the *unique* witness $w(s)$ assigned to $C(s)$. However, the LPs in AdjacencyOracle in ℓ-RS have a special structure and generally produce different witness points for the same cell. By contrast, LPs in AdjacencyOracle of m-RS are exactly in the form (2.1) and produce the unique witness point directly.

5. The m-IE version. In this section we modify FlIE to be directly comparable with m-RS. We begin with several remarks on 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. Steps {f4} to {f6} comprise 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. We will see in section 5.4 that this filter is more efficient. In particular, FlIE needs no ParentSearch procedure—the uniqueness of each sign vector that is examined is ensured by the 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*. Nevertheless, a smoother analysis taking into account what exactly happens inside the $\mathrm{lp}(\cdot)$ factor (which is treated as a black box here) would be desirable.

5.2. New cell ⇒ some additional LPs. Observe the following behavior of the algorithms: if a new cell is found during their run, some additional LPs are to be solved. This will be formalized in Proposition 5.2. First we introduce some notation in Definition 5.1. We stress that the order in which cells are enumerated is influenced by the LP solver for the problem (2.1). The ordering of generators and the choice of the root cell can also influence the enumeration. Thus, the notions of RS- and IE-level from the upcoming definition should be understood relatively to these factors.

DEFINITION 5.1. *For a sign vector s^m, we define*

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

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

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 10/20/22 to 128.93.162.241 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

PROPOSITION 5.2.

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

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

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

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

Proof.

(i) RevSearchEnu in its 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 a cell enumeration process find a new witness point and some do not. We accordingly call these LPs *successful* and *unsuccessful*, respectively. In IncEnu, a new witness point means a new cell. In m-RS, the cell corresponding to the new witness point must still be checked by ParentSearch.

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

To support the claim that IncEnu performs better than m-RS, one should prove that it is not possible for the average RS-level to be lower than the average IE-level. The modification of FlIE to m-IE is necessary just to facilitate comparison of the number of unsuccessful LPs. The main problem is that Reverse Search tests only PM-flips, while IncEnu must sometimes solve LPs corresponding to MP-flips, too. This obstacle will have to be overcome in proving the 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 in step {f6}. Every successful LP yields a witness point, say w, and results in the output of a new cell. Thus $f_{\text{IE}} = |\mathcal{C}_m|$.

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 neighboring cells with sign vectors s, s'. Assume that this facet is in the hyperplane h_j. Then the flip of the jth sign of s or s' is successful. Either s_j or s'_j must be "+"; say it is s_j. Then RevSearchEnu(s) must solve the LP testing whether the flip of the jth sign of s is successful. Thus 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 the number of facets of $\mathcal{A}_m$.

Downloaded 10/20/22 to 128.93.162.241 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Algorithm 4. m-IE algorithm. Global variables w and s are required; initially, w is assumed to be an interior point of $C(r)$ for $r = (1, \dots, 1)^T$.

```
{1}  Function m-IE(i):
{2}    output fullSized(w)
{3}    for each j := i + 1, ..., m:
{4}      s := (s, -1)
{5}      if there 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)
```

Since the rank of G is d, every cell has at least d neighbors, 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 everything together, we obtain

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

Note that the lower bound d on the number of neighbors of a cell can be attained only for unbounded cells. For bounded cells, the bound is $d + 1$.

5.4. Formulation of m-IE: Dealing with unsuccessful LPs. Here we modify FlIE in such a way that it solves at most as many unsuccessful LPs as m-RS does. 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 in positions $i + 1, \dots, m$ are tested. Hence, it would suffice if we could satisfy the following in m-IE:

$$\begin{aligned} &\text{when } m = \text{IE}(i) \text{ is called,} \\ &\text{all signs of fullSized}(w) \text{ in positions indexed } \geq i + 1 \text{ are 1,} \end{aligned} \tag{5.4}$$

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

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

We assume without loss of generality that the m-IE algorithm starts in the cell with sign vector $r = (1, \dots, 1)^T$. Two global variables are initialized: $w = w(r)$ as an interior point of $C(r)$, and a sign vector $s = s^0$ as the empty sign vector. With this initialization property (5.4) is satisfied for m-IE(0).

Now suppose that we run m-IE(i) such that property (5.4) is satisfied. First, the full-sized sign vector s^m corresponding to w is found and output. Then, during the **for** cycle {3} of m-IE, extensions s^j of s^i are tested for $j = i + 1, \dots, m$. For example, consider the case $j = i+1$. If sign vector $(s^i, -1)$ generates a cell of A_{i+1} (steps {4}–{5}), we take its witness ω and consider the line segment (w, ω). We examine the number of hyperplanes it intersects. Let $\sigma \in \{\pm 1\}^m$ denote the sign vector such that $\omega \in C(\sigma)$. The number of intersected hyperplanes can be measured by the Hamming

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 10/20/22 to 128.93.162.241 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

distance $\operatorname{dist}(s^m, \sigma) := |\{k : s^m_k \neq \omega_k\}|$. 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 $\operatorname{dist}(s^m, \sigma) = 1$, then σ differs from s^m exactly in the jth sign and there are no other "$-$" signs after the jth sign. So let us turn to the case $\operatorname{dist}(s^m, \sigma) \geq 2$. Let k and l be the indices of the first two hyperplanes intersected by line segment (w, ω); in case of ambiguity, we use symbolic perturbation. Let α_k and α_l denote the intersections of line segment (w, ω) 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 that is a witness point of $\operatorname{flip}_j(s^m)$, and furthermore ω now corresponds to a new full-sized sign vector σ, for which $\operatorname{dist}(s^m, \sigma) = 1$ (step {7}). This is due to the fact that the line segment (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 run ensures that property* (5.4) *holds for every m-IE call.*

Problems in swapping generators. Interchanging generators brings some complications.

(i) The generator list changes during the run of the algorithm; one must track these changes and ensure the correctness of the output.

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

There is a natural solution to (i), similar to the idea of AdjacencyOracle in 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 from the recursion we put the last generator back onto the list of used generators. If necessary, each generator can hold its initial ID, and in the output phase we can sort the generators back into their 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 various solutions to (ii). The witness points along a branch of the recursion tree can be stored in memory as rational vectors at the cost of an increase in space complexity to $O(m \operatorname{lp}(G))$. Alternatively, we can represent a witness by the set of optimal basic indices of the corresponding LP. When storing m witness points, this is at the cost of additional $O(md)$ memory. This solution is the most efficient, since $O(md)$ is dominated by $O(\operatorname{lp}(G))$. Alternatively, a witness point can be forgotten and then recomputed from scratch as in {10}. This is at the cost of $O(|\mathcal{C}_m|)$ additional LPs. We have used this version in Algorithm 4 just for simplicity of presentation (there is no asymptotic difference). We also assume this implementation in Theorem 5.6.

5.5. Overall properties of m-IE.

THEOREM 5.5. *The m-IE algorithm has time complexity $\mathcal{O}(|\mathcal{C}_m|\ m\ \operatorname{lp}(m, d)))$ and space complexity $\mathcal{O}(\operatorname{lp}(m, d))$.*

THEOREM 5.6. *Suppose the sign vector $s^m = (1, \ldots, 1)^T$ is feasible, that is, $C(s^m)$ is a cell of $\mathcal{A}_m$. Let λ_{IE} and λ_{RS} stand for the number of LPs solved during the running of m-IE and m-RS, respectively, when started from s^m. Then*

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

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 10/20/22 to 128.93.162.241 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

Proof. The distinction between successful and unsuccessful LPs was introduced in section 5.3. By Lemma 5.3, the number of successful LPs solved by IE (and FlIE) is at least $d/2$ times better compared to the number of successful LPs for m-RS. This contributes $|\mathcal{C}_m|(\frac{d}{2}-1)$ to the term subtracted. 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 $|\mathcal{C}_m|$. This accounts for the other $|C_m|$ that is subtracted from λ_{RS}.

Every unsuccessful LP solved during the run 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 a similar background to the construction of m-IE from section 5. The advantage of ℓ-IE is that we will solve smaller LPs. The basic idea lies in combining 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 hyperplanes that are neighboring with respect to the current cell. Since the maximal number ℓ of neighboring 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. For details see section 4.4.

The consecutive building of the set of neighboring 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 is found—and this procedure of finding the nearest hyperplane is the same for both ℓ-AO and m-IE.

The difference is that m-IE(i) always includes the halfspaces of hyperplanes $h_1, \ldots, h_i$ in the LPs. On the other hand, ℓ-RS(s^m) computes from scratch (in the worst case) whether these hyperplanes are neighboring and includes them only if necessary. 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. To make m-IE comparable with ℓ-RS, we try to test the flips of interesting signs (after the ith sign when m-IE(i) is called), and add the hyperplanes $h_1, \ldots, h_i$ 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 AdjacencyOracle in ℓ-RS), as follows.

- A 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).
- A list N of generators, corresponding to the neighboring hyperplanes found so far.

In ℓ-IE, we no longer use a global variable for a sign vector. For the sake of simplicity, we use one sign vector at 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 at every level

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 10/20/22 to 128.93.162.241 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

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

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

of recursion. The global variable w for a witness point is used and initialized in the same way as in the case of m-IE.

The ℓ-IE algorithm iterates the **while** cycle {4} until all elements of U are examined. In each iteration, ℓ-IE selects an arbitrary index $j \in U$. Recall that in (4.1) we defined the set

$$E(N, s^m, j) = \{\xi \in \mathbb{R}^d : \ \forall i \in N \ s_i^m(g_i^T\xi - 1) \geq 0, \ g_j^T\xi \leq 1\}.$$

Step {6} of ℓ-IE tests whether $E(N, s^m, j)$ passes test (2.1), that is, whether there exists a witness point ω of $E(N, s^m, j)$. We take the halfspaces induced by the neighboring 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 first and (if available) the second nearest hyperplanes h_k and h_l along the ray (w, ω), together with the corresponding intersection points α_k and α_l. One of two cases may then occur.

- The index k is in the set U of unexamined hyperplanes. Then we compute a new interior point w from α_k and α_l (if available) and swap the generators g_k and g_j. As a result, we know that h_j is the neighboring hyperplane for $C(s^m)$ and we can call ℓ-IE$(j, \text{flip}_j(s^m))$ in step {12}. The property (5.4) is satisfied. After returning from the recursion, we must reset the interior point w in the same way as m-IE does (step {13}).
- The index k is not in the set of unexamined hyperplanes. The flip of the kth sign need not be examined; we just add it to the set of neighboring hyperplanes (step {15}).

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

Recall the definition of ℓ as the maximum number of neighboring cells among all cells of $\mathcal{A}_m$.

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 10/20/22 to 128.93.162.241 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

Downloaded 10/20/22 to 128.93.162.241 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

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 without loss of generality generates a cell. Denote by λ^m_{IE} and λ^m_{RS} the number of $m \times d$ LPs solved during the run of ℓ-IE and ℓ-RS, respectively, beginning with interior point w and sign vector r. Then*

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

Furthermore, let $\lambda^\ell_{\mathrm{IE}}$ and $\lambda^\ell_{\mathrm{RS}}$ denote the number of LPs of size at most $\ell \times d$ solved during the run of ℓ-IE and ℓ-RS, respectively, beginning with the sign vector r with witness point w. Then

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

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

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

7. Computational experiments. In this section we provide some initial empirical comparison of the algorithms we have presented. The experiments appear to give evidence for the following conjecture.

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

As well as evidence for the above conjecture, we also used the experiments to shed some light on the following general questions.

(Q1) How much better are incremental enumeration algorithms than reverse search algorithms?
We compare the 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 the "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, which is surely the most time-consuming (and most frequently called) subprocedure. The number of LPs is the most natural comparison criterion. Recall that we also used it in the asymptotic bounds in sections 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 no smaller than the sizes of LPs in incremental enumeration algorithms. We also record elapsed times.

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

Implementation. We implemented the algorithms in Python 2.7. We used the `Gurobi` solver (see [7]) for solving LPs.

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 10/20/22 to 128.93.162.241 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

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

Nondegenerate and degenerate arrangements. An arrangement determined by $g_1, \ldots, g_m \in \mathbb{R}^d$ is *nondegenerate* if no d-tuple of vectors $g_1, \ldots, g_m$ is linearly dependent and no $(d+1)$-hyperplanes intersect at a common point, otherwise it is *degenerate*. In other words, the arrangement is nondegenerate if $g_1, \ldots, g_m$ are in a general position. A nondegenerate arrangement has the maximum possible number of cells.

A brief description of the test classes of arrangements follows.

Randomly generated arrangements. Nondegenerate 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, nondegeneracy of the arrangements was tested ex post using the number of cells enumerated. We used the fact that the upper bound on the number of cells is attained for nondegenerate arrangements.)

Degenerate random arrangements were generated in a similar way. The amount of degeneracy was controlled with a *degeneracy ratio* r. First, d random generators were generated. Second, each subsequent 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 $g_1, \ldots, g_{i-1}$; with probability $1 - r$, g_i was chosen at random.

In the summary of results these instances are denoted by the triple "m, d, r."

Arrangements combinatorially dual to permutahedra. Arrangements dual to permutahedra are highly degenerate and have the property that every cell has d neighboring cells, that is, 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" in Table 1.

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

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

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

Observation 7.2. Assuming $i \ge d$, 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, the arrangement $\mathcal{A}_{d-2}$ has 2^{d-2} cells. Second, all the 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 $\operatorname{rank}(G'_i) = 2$ and no two generators in G'_i are linearly dependent, the number of its cells attains the bound (1.1) for $d = 2$ and $m = i - d + 2$. The arrangement G'_i has $2(i - d + 2)$ cells. ∎

LEMMA 7.3. *The number of LPs solved by IncEnu during cell enumeration of* $\mathcal{A}_m$ *with generators* (7.1) *is*

$$m\, 2^{d-1}(m - d + 2) - \left(\sum_{i=0}^{d-2} 2^i i + \sum_{i=d-1}^{m} 2^{d-1} i \right). \tag{7.2}$$

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 10/20/22 to 128.93.162.241 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

TABLE 1

Experimental results. For each algorithm, the left-hand number is #LPs; the right-hand number is elapsed time. The last column is the ratio of #LPs for m-RS to #LPs for IncEnu.

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

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 10/20/22 to 128.93.162.241 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

Proof. According to Proposition 5.2, the number of LPs solved by IncEnu equals $m\,|\mathcal{C}_m|$ reduced by the sum of IE-levels of all cells, which is exactly the formula (7.2). For $d-1 \le j \le m$, at the jth 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. The number of cells with IE-level k such that $1 < k < d-1$ is easily calculated, since each h_k doubles the number of cells of $\mathcal{A}_{k-1}$. ∎

COROLLARY 7.4. *For every fixed d and $m \ge d-1$, 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

$$\begin{aligned}
&\frac{m\,|\mathcal{C}_m| - \left(\sum_{i=0}^{d-2} 2^i i + \sum_{i=d-1}^{m} 2^{d-1} i\right)}{|\mathcal{C}_m|} \\
&\quad = m - \frac{2^{d-1}(d-2) - 2^{d-1} + 2 + 2^{d-1}(m+d-1)(m+2-d)/2}{2^{d-1}(m-d+2)} \\
&\quad = m - \frac{m+d-1}{2} - \frac{d-3}{m-d+2} - \frac{1}{2^{d-2}(m-d+2)} = I(m).
\end{aligned}$$

Since $I(m) \to \frac{m}{2}$ as $m \to \infty$, the total number of LPs is $\Omega(|C_m|\,m)$. ∎

These arrangements have one complex $(d-2)$-dimensional face, usually called a *ridge*; in Table 1 they are denoted 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. The ratio "#LPs of m-RS/#LPs of IE" in the last column of the table leads us to the formulation of Conjecture 7.1.

Several observations about the behavior of the algorithms can be made.

- The m-versions of the algorithms seem to perform better than the 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. Therefore 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 such a way that no information about the LP previously solved is reused by the solver, ℓ-RS becomes better than m-RS and 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 an implementation issue. The answer to (Q2) is then that the m-versions of the algorithms are faster.
- We proved in section 5.4 (for m-IE) and section 6 (for ℓ-IE) that incremental enumeration algorithms perform better than reverse search algorithms. Additionally, in nondegenerate or slightly degenerate cases, incremental enumeration algorithms seem to be $\mathcal{O}(d)$ times better. This is what we have stated as 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 a lower number of unsuccessful LPs in general.

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

8. Conclusions. We have 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 $\mathcal{O}(|\mathcal{C}_m|\, m\, \mathrm{lp}(G))$ 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 those versions of Reverse Search. In fact, we have shown that, regardless of which version of Reverse Search one chooses, our algorithm performs better (Theorems 5.6 and 6.1(b)).

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

Acknowledgments. The help of Andrew Goodall from the Faculty of Mathematics and Physics of Charles University and three anonymous referees is highly acknowledged. Their suggestions and observations significantly contributed to many improvements made to the paper during the editorial process.

REFERENCES

[1] K. Allemand, K. Fukuda, T. M. Liebling, and E. Steiner, *A polynomial case of unconstrained zero-one quadratic optimization*, Math. Program., 91 (2001), pp. 49–52, doi:10.1007/s101070100233.

[2] D. Avis and K. Fukuda, *Reverse search for enumeration*, Discrete Appl. Math., 65 (1996), pp. 21–46, doi:10.1016/0166-218X(95)00026-N.

[3] R. C. Buck, *Partition of space*, Amer. Math. Monthly, 50 (1943), pp. 541–544, doi:10.2307/2303424.

[4] M. R. Bussieck and M. E. Lübbecke, *The vertex set of a* 01*-polytope is strongly P-enumerable*, Comput. Geom., 11 (1998), pp. 103–109, doi:10.1016/S0925-7721(98)00021-2.

[5] H. Edelsbrunner, J. O'Rourke, and R. Seidel, *Constructing arrangements of lines and hyperplanes with applications*, SIAM J. Comput., 15 (1986), pp. 341–363, doi:10.1137/0215024.

[6] J.-A. Ferrez, K. Fukuda, and Tm. M. Liebling, *Solving the fixed rank convex quadratic maximization in binary variables by a parallel zonotope construction algorithm*, European J. Oper. Res., 166 (2005), pp. 35–50, doi:10.1016/j.ejor.2003.04.011.

[7] *Gurobi Optimizer Reference Manual*, Gurobi Optimization, Inc., Houston, TX, 2014.

[8] Tm. Ottmann, S. Schuierer, and S. Soundaralakshmi, *Enumerating extreme points in higher dimensions*, in STACS 95: 12th Animal Symposium on Theoretical Aspects of Computer Science, Lecture Notes in Computer Science 900, Springer, Berlin, 1995, pp. 562–570, doi:10.1007/3-540-59042-0_105.

[9] N. Sleumer, *Output-sensitive cell enumeration in hyperplane arrangements*, in Algorithm Theory—SWAT'98: 6th Scandinavian Workshop on Algorithm Theory, Lecture Notes in Computer Science 1432, Springer Berlin, 1998, pp. 300–309, doi:10.1007/BFb0054377.

[10] T. Zaslavsky, *Facing up to Arrangements: Face-Count Formulas for Partitions of Space by Hyperplanes*, Memoirs of the American Mathematical Society 154, American Mathematical Society, Providence, RI 1975.

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 10/20/22 to 128.93.162.241 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy