Counting External Facets of Simple Hyperplane Arrangements

FENG XIE

January 7, 2008

Department of Computing and Software
McMaster University
Hamilton, Ontario, Canada
xief@mcmaster.ca

Abstract: The number of external facets of a simple arrangement depends on its combinatorial type. A computation framework for counting the number of external facets is introduced and improved by exploiting the combinatorial structure of the set of sign vectors of the cells of the arrangement.

1 Background and introduction

n hyperplanes in dimension d form a hyperplane arrangement. An hyperplane arrangement is called simple if $n \geq d$ and any d hyperplanes intersect at a unique distinct point. A facet of a hyperplane arrangement belongs to either zero, one or two bounded cells. We call a facet external if it belongs to exactly one bounded cell. It has been shown that the line arrangement ($d = 2$) that minimizes the number of external facets maximizes the average diameter, but it is not known whether this relation holds for general dimension. The computational results for small n and d will give us a better insight into this problem. One of the computational combinatorial problems herein is as following.

Given a simple arrangement $\mathcal{A}_{d,n}$ represented by n inequalities with d variables, count its external facets.

As an example, Figure 1 shows the enumeration of the 6 line arrangements formed by 5 lines, or $\mathcal{A}_{2,5}$. Among the 6 arrangements, $\mathcal{A}^o$ has 8 external facets, which is minimal. The star-shaped arrangement at the bottom right corner has 10 external facets and all the others have 9.

Let $\mathcal{A}_{d,n}$ be a simple arrangement formed by hyperplanes $h_1, h_2, \ldots, h_n$. Each hyperplane partition the space into 2 sides: positive and negative. By giving each hyperplane of $\mathcal{A}_{d,n}$ an orientation indicating which side is positive, each cell is associated with a sign vector whose ith element ($i = 1, \ldots, n$) indicates which side of hyperplane h_i the cell is located on ($+$ and $-$ for positive and negative side respectively). See Figure 2 for the sign vectors of all the cells of $\mathcal{A}^o_{2,4}$.

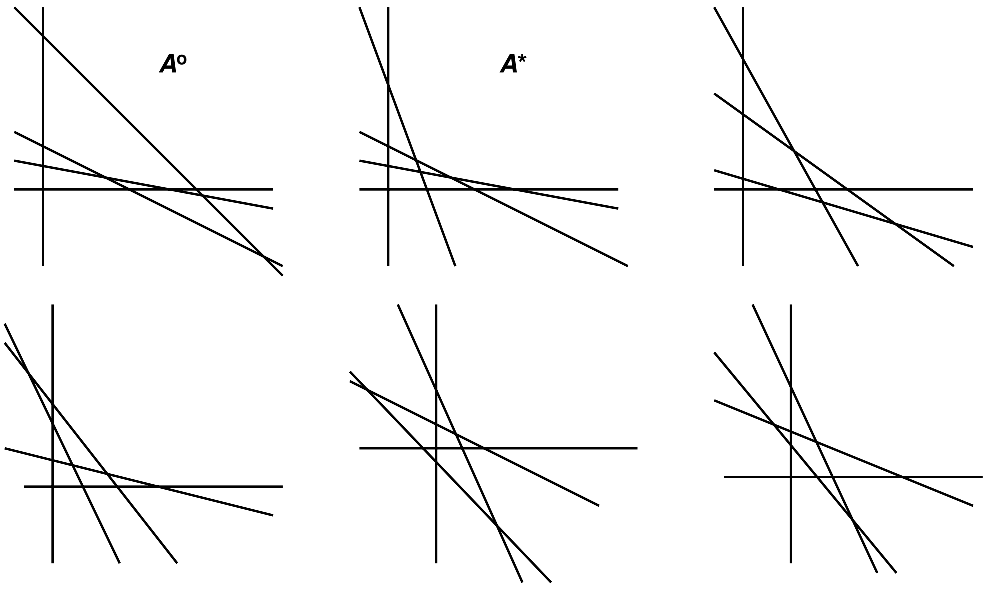

Figure 1: Enumeration of $\mathcal{A}^o_{2,5}$

2 Existing algorithm

To solve the problem, the following algorithm is presented in my Masters thesis [5].

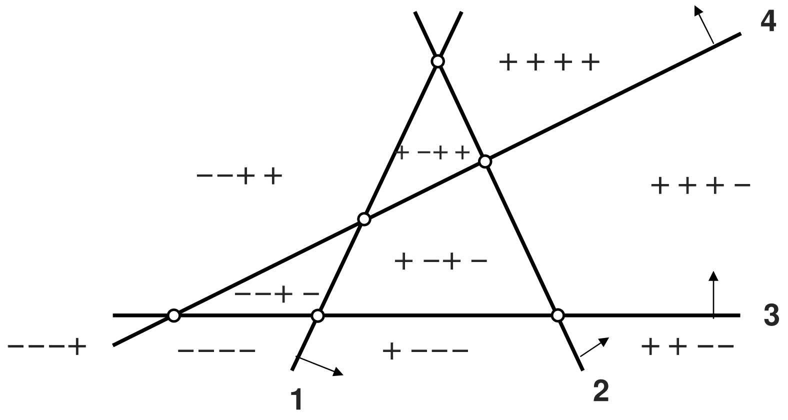

Figure 2: Sign vectors of the cells of $\mathcal{A}^o_{2,4}$

```
Algorithm 1: CountExternalFacets
    input  : $\mathcal{A}_{d,n}$
    output: $f_e$ (number of external facets)
1   Initialize $f_e$;
2   $\mathcal{A}_{d+1,n} \leftarrow$ DimensionLift ($\mathcal{A}_{d,n}$);
3   SignVectors $\leftarrow$ EnumerateCells ($\mathcal{A}_{d+1,n}$);                /* minksum */
4   foreach SignVector $\in$ SignVectors do
5       if IsBounded ($\mathcal{A}_{d,n}$, SignVector) then
            /* get the indices of hyperplanes that tightly bound the cell      */
6           IND $\leftarrow$ RemoveRedundancy ($\mathcal{A}_{d,n}$, SignVector);
7           foreach ind $\in$ IND do
                /* invert ind'th sign of SignVector to get the neighboring cell  */
8               SignVectorNeighbor $\leftarrow$ Invert (SignVector, ind);
9               if IsBounded ($\mathcal{A}_{d,n}$, SignVectorNeighbor) then
10                  $f_e \leftarrow f_e + 1$;
11  return $f_e$;
```

Although the algorithm works for small instances, there is still much room to improve. Being modeled as linear programming and computational geometry problems respectively, the subroutines *RemoveRedundancy* in line 6 and *IsBounded* in line 9 are the most expensive steps in the loop.

Given a sign vector of a cell, we have n inequalities that define the cell: $\{\mathbf{a_i}^T\mathbf{x} \leq c_i,\ i = 1,\ldots,n\}$. Some of the inequalities might be redundant. To determine whether the ith inequality ($\mathbf{a_i}^T\mathbf{x} \leq c_i$) is redundant, we can solve the following linear programming problem first.

$$\begin{cases} \max & \mathbf{a_i}^T\mathbf{x} & & \\ s.t. & \mathbf{a_1}^T\mathbf{x} & \leq & c_1, \\ & & \cdots & \\ & \mathbf{a_i}^T\mathbf{x} & \leq & c_i + 1 \\ & & \cdots & \\ & \mathbf{a_n}^T\mathbf{x} & \leq & c_n \end{cases}$$

Let the solution of the problem be f^*. Then $\mathbf{a_i}^T\mathbf{x} \leq c_i$ is redundant if $f^* \leq c_i$. It is known that a simple arrangement $\mathcal{A}_{d,n}$ has $\binom{n-1}{d}$ bounded cells, and for each bounded cell, there are n linear programming problems to solve. Therefore, we have to solve $n\binom{n-1}{d}$ linear programming problems in order to remove all the redundant inequalities.

Given a cell of $\mathcal{A}_{d,n}$ represented by n inequalities, checking its boundedness is equivalent to the polyhedron vertex enumeration problem, a well studied computational geometry problem. An algorithm presented in [1] has an output sensitive running time of $O(ndv)$, where v is the number of vertices of the cell. It is known that a simple arrangement $\mathcal{A}_{d,n}$ has $\sum_{i=0}^{d}\binom{n}{i}$ cells [3], which means that we have to solve $O(n^d)$ vertex enumeration problems.

As we can see, in this algorithm, the running time spent in redundancy removal and boundedness check explodes as n and d increases. Moreover, numerical difficulties prevent the algorithm from giving correct results in certain cases.

3 Combinatorial approaches

the set of sign vectors of the cells, corresponding to an oriented matroid [2], is a combinatorial abstraction of arrangements [4]. Thus, it is possible to tackle the problems of redundancy removal and boundedness check by purely combinatorial approaches, which are more efficient and free of numerical difficulties.

3.1 Redundancy removal

The combinatorial approach to redundancy removal is based on the following self-evident fact.

Fact 1 *Two cells share a facet if and only if their sign vectors differ by one sign.*

Each equality corresponds to a hyperplane where the facets reside. Given a cell P, if an inequality is redundant, i.e. the corresponding hyperplane does not tightly bound the cell, then any cell on the other side of the hyperplane has a sign vector with at least 2 signs different from the sign vector of P. Otherwise, by Fact 1 there exists a cell on the other side of the hyperplane that shares a facet with P, which is not possible because P is not tightly bounded by the hyperplane. Therefore, we have the following fact and new algorithm for redundancy check.

Fact 2 *Given the sign vector of a cell of an arrangement $\mathcal{A}$, a hyperplane h_i of $\mathcal{A}$ is redundant if the sign vector obtained by negating the sign corresponding to h_i does not correspond to a cell of $\mathcal{A}$.*

Algorithm 2: RemoveRedundancy

```
input  : SignVector (of a cell), SignVectors (of all cells)
output: IND (indices of nonredundant inequalities)
1 IND ← $\phi$;
2 for $i \leftarrow 1$ to $n$ do
      /* invert i'th sign of SignVector to get the neighboring cell    */
3     SignVectorNeighbor ← Invert (SignVector, $i$);
4     if SignVectorNeighbor ∈ SignVectors then
5         IND ← IND ∪{$i$};
6 return IND;
```

Obviously, Algorithm 2 drastically improves the efficiency of redundancy removal and is immune from numerical problems.

3.2 Boundedness check

As described in Section 6.2 of [5], given a simple arrangement $\mathcal{A}_{d,n}$ the sign vectors of the cells of $\mathcal{A}_{d,n}$ are obtained by first converting $\mathcal{A}_{d,n}$ into a linear arrangement $\mathcal{A}_{d+1,n}$ using dimension lifting (see Figure 3). This way, each cell of $\mathcal{A}_{d+1,n}$, which is an origin-pointed cone, corresponds to a cell of $\mathcal{A}_{d+1,n}$. To remove symmetry, we extend $\mathcal{A}_{d+1,n}$ to $\mathcal{A}_{d+1,n+1}$ by adding a hyperplane h_{n+1} with equation $x_{d+1} = 0$, where x_{d+1} is the coordinate variable of the new dimension. Only the dimension-lifted cells on one side of h_{n+1} need to be studied. More importantly, h_{n+1} also helps to identify the bounded cells of $\mathcal{A}_{d,n}$: It is not hard to see that a cell of $\mathcal{A}_{d,n}$ is bounded if and only if h_{n+1} tightly bounds the corresponding dimension-raised cell of $\mathcal{A}_{d+1,n+1}$, i.e., h_{n+1} is nonredundant. This way, the boundedness check problem is transformed into a redundancy check problem in higher dimension, which has been discussed in the previous section.

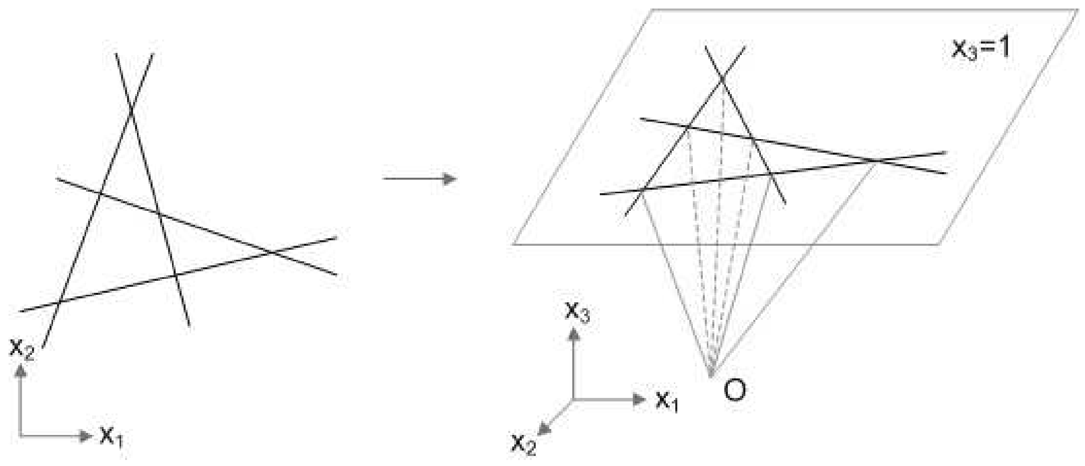

Figure 3: Lifting an arrangement from dimension 2 to dimension 3.

4 Implementation

All the algorithms are implemented using *Python*, a scripting language influenced by *Perl.* As a “glue language”, its strong text processing capability makes it ideal for this project, in which most of the computation extensive tasks are taken over by existing softwares (CDD, minksum) with different input/output formats. Additionally, Python’s loose syntax and rich external modules are quite convenient.

As far as the graph is concerned, I turn to *NetworkX*, a third-party *Python* module that handles graph representation and common graph subroutines.

References

[1] D. Avis and K. Fukuda, A pivoting algorithm for convex hulls and vertex enumeration of arrangements and polyhedra, Discrete Computational Geometry, **8**:295-313 (1992).

[2] A. Bjöner, M. L. Vergnas, B. Sturmfels, N. White and G. M. Ziegler, Oriented Matroids, Cambridge University Press (1993).

[3] H. Edelsbrunner, Algorithms in Combinatorial Geometry, Springer-Verlag (1987).

[4] L. Finschi and K. Fukuda, Combinatorial generation of small point configurations and hyperplane arrangements, Manuscript.

[5] F. Xie, Hyperplane Arrangements with Large Average Diameter, PhD Thesis, McMaster University (2007).