

A Note on Point Location in Arrangements of Hyperplanes

DING LIU

Department of Computer Science, Princeton University,
Princeton, NJ, 08544 USA
dingliu@cs.princeton.edu

Abstract

We give an algorithm for point location in an arrangement of n hyperplanes in E^d with running time $\text{poly}(d, \log n)$ and space $O(n^d)$. The space improves on the $O(n^{d+\epsilon})$ bound of Meiser's algorithm [5] that has a similar running time.

Keywords: Computational Geometry, Point Location, Hyperplane Arrangements

1 Introduction

We consider point location in arrangements of hyperplanes. The problem is to preprocess a collection H of n hyperplanes in E^d so that given a query point q , we can quickly identify the cell of the arrangement of H whose closure contains q . The trivial point location algorithm (without preprocessing) runs in time $O(dn)$, by testing the incidence relationship (on, above, or below) between q and every hyperplane of H . Clarkson [3] presented an algorithm whose running time is $O(\log n)$ (throughout this paper \log refers to logarithm to the base 2, unless another base is given explicitly). His algorithm uses space $O(n^{d+\epsilon})$, where $\epsilon > 0$ could be made arbitrarily small. Chazelle [1] improved the space complexity to $O(n^d)$ which is asymptotically optimal.¹ In both cases, the running time is exponential in d . Meiser [5] presented a solution with $\tilde{O}(d^4 \log n)$ running time² and $O(n^{d+\epsilon})$ space for arbitrary $\epsilon > 0$. His algorithm is interesting in that it is the first algorithm for point location in arrangements of hyperplanes with running time polynomial both in d and $\log n$. However the space of Meiser's algorithm is not optimal.

In this paper we show that by combining Chazelle's technique with Meiser's algorithm, one can obtain another algorithm whose running time remains to be polynomial in d and $\log n$, and it uses asymptotically optimal space. Thus the new algorithm shares the virtues of the previous two. Specifically our new algorithm runs in time $\tilde{O}(d^5)(\log n + d)$ and it uses space $O(n^d)$. The big- O

¹This is because the worst-case combinatorial complexity of the arrangement of n hyperplanes in E^d is $\Theta(n^d)$.

²The $\tilde{O}()$ notation hides a factor of $\log^{O(1)} d$. Meiser claimed a $O(d^5 \log n)$ bound but more precisely it is $\tilde{O}(d^4 \log n)$.

in our space bound (and in all the above mentioned algorithms) hides a factor that depends only on d . But this is an exponential dependence. The justification for this is that the n^d factor will appear anyway, and it subsumes other factors that depend only on d . For this reason throughout this paper in the space bound we always hide factors that depend only on d . On the other hand in the time bound d is regarded as a variable and it is always stated explicitly.

2 Previous Algorithms

Given a set H of hyperplanes in E^d , we use $\mathcal{A}(H)$ to denote the arrangement of H , and we use $\mathcal{T}(H)$ to denote a triangulation of $\mathcal{A}(H)$ [2, 3]. All previous algorithms [3, 1, 5] for point location in hyperplane arrangements are based on the use of a $(1/r)$ -cutting (though in [3, 5] “cutting” is not explicitly defined). A $(1/r)$ -cutting [4] for a set of n hyperplanes in E^d is a collection of possibly unbounded d -dimensional closed simplices with disjoint interiors, such that they together cover E^d , and the interior of each simplex intersects at most n/r hyperplanes. The algorithms of Clarkson and Meiser rely on the following result [3, 5]: For any set H of n hyperplanes in E^d and any $1 < r \leq n$, there exists a set $R \subset H$ of $O(d^2 r \log^2(dr))$ hyperplanes such that $\mathcal{T}(R)$ is a $(1/r)$ -cutting for H . This immediately leads to the following divide-and-conquer approach for point location in $\mathcal{A}(H)$: Construct a $(1/r)$ -cutting $C = \mathcal{T}(R)$ for H for some small constant r (say $r = 2$), and identify the simplex $s \in C$ that contains the query point q by locating q in C . Let H_s denote the set of hyperplanes of H that intersect s . By the definition of C , $|H_s| \leq n/r$. Then it suffices to locate q in $\mathcal{A}(H_s)$. This is again a point location problem, but of smaller size. We can thus solve the original problem recursively.

To implement this recursive algorithm, a tree is built in preprocessing time. We use C_k ($k \geq 0$) to denote a $(1/r^k)$ -cutting for H . Each node on the k -th level of this tree (with root on level 0) is a simplex of C_k . To build the tree, we start from E^d and view it as a unbounded simplex that forms C_0 . It is the root of the tree. For $k > 0$ we refine C_{k-1} into C_k as follows: For each simplex $s \in C_{k-1}$ we compute a $(1/r)$ -cutting for H_s . The simplices of this cutting are called the contributions of s to C_k , and they form the children of s in the tree. We keep splitting each simplex s in the tree until $|H_s| \leq r$, where we stop and make s a leaf. The tree constructed this way has depth $\lceil \log_r n \rceil$.

Locating the query is easy: We walk down the tree from its root, and at each internal node s branch to the child of s that contains the query. Upon reaching a leaf, we use the trivial algorithm to locate the query with respect to at most r hyperplanes that intersect the simplex at that leaf. The insight of [5] is that each branching step at an internal node can be done in $O(d^2 t)$ time using $O(t^{2d})$ space where $t = O(d^2 r \log^2(dr)) = \tilde{O}(d^2)$, provided that a canonical triangulation [3, 5] is used to compute the $(1/r)$ -cutting for each H_s . This observation makes the query time in [5] $\tilde{O}(d^4 \log n)$. The space $S(n)$ in [5] satisfies the recursion: $S(n) = O(t^{2d}) + O(t^d)S(n/r)$ for $n > r$; and $S(n) = O(r^d)$ for $n \leq r$. This recursion solves to $S(n) = n^{d+\epsilon}$, where $\epsilon > 0$ could be made arbitrarily small by adjusting r .

Chazelle [1] improved the space (that is, the size of the tree) to $O(n^d)$ by refining C_{k-1} into C_k in a more subtle way. His idea is that it is unnecessary to compute a $(1/r)$ -cutting for every simplex $s \in C_{k-1}$. Instead, the size of the cutting for s should be adaptive to $v(H_s, s)$, the number of vertices of $\mathcal{A}(H_s)$ inside s . In particular Chazelle refines a simplex $s \in C_{k-1}$ as follows: Pick

a large enough constant r_0 that depends on d , if $|H_s| \leq n/r_0^k$ then s stands as such; otherwise, let $\rho_0 = |H_s|r_0^k/n$. First we compute a $1/(2d\rho_0)$ -approximation A_s for H_s , then we compute a $1/(2d\rho_0)$ -net R_s for A_s such that R_s is sparse for s . See [1] and the references therein for the meaning of $(1/2d\rho_0)$ -approximation and sparse $(1/2d\rho_0)$ -net, and other related definitions. The contribution of s to C_k is the set of simplices obtained by canonically triangulating the portion of $\mathcal{A}(R_s)$ inside of s . In [1] it is proved that $|C_k| = r_0^{(k+1)d}$ for r_0 large enough that depends on d but not on r_0 and k . To do point location, we use Chazelle's cutting to build a tree as described above. The height of this tree is set to $h = \lceil \log_{r_0} n \rceil$ so that every simplex at a leaf is fully enclosed in a cell of $\mathcal{A}(H)$. The size of the tree is $O(|C_h|) = O(n^d)$.

3 The New Algorithm

We note that by combining the algorithm of Chazelle with that of Meiser we can obtain a new algorithm that shares the virtues of the previous two. We build the search tree, called the *main tree*, by following Chazelle's cutting, and we use Meiser's algorithm to navigate in the main tree. In particular, we use Meiser's algorithm to do branching at each internal node. In [1] the dimension d is regarded as a constant and is usually hidden by the big- O notation (except when d appears on the exponent). However it is not hard to go through the Equations, Lemmas and Theorems in [1] and reveal, for each quantity we are interested in, its dependence on d . Specifically we can work out that, in the following recursive equation on page 153 of [1]:

$$|C_k| \leq c \left(\frac{r_0^k \log r_0}{n} \right)^d n^d + cr_0^{d-1} (\log r_0)^d |C_{k-1}|,$$

the parameter c is $d^{O(d)}$, and r_0 has to satisfy $r_0 > c(\log r_0)^d$ for $|C_k|$ to be solved to $r_0^{(k+1)d}$. This shows that we can let r_0 be $d^{\Theta(d)}$.

We adopt notations from the last section. Recall for each simplex s in Chazelle's cutting, it is refined according to the arrangement of at most $|R_s|$ hyperplanes. Again it is easy to reveal the dependence of $|R_s|$ on d in [1]: $|R_s| = O(d^2 r_0 \log(dr_0)) = d^{O(d)}$. Now the idea is that for each s we build Meiser's data structure on R_s in preprocessing time, and use this structure to branch to the child of s that contains the query. But there is a problem: Meiser's algorithm returns a cell of $\mathcal{A}(R_s)$, while we are seeking for a finer answer, namely a simplex of $\mathcal{T}(R_s)$. To deal with this problem, we view $\mathcal{T}(R_s)$ as part of $\mathcal{A}(R_s^0)$ where R_s^0 is a minimal set of hyperplanes such that each hyperplane in R_s^0 supports at least one $(d-1)$ -dimensional facet of $\mathcal{T}(R_s)$. Since each cell of $\mathcal{A}(R_s^0)$ is fully enclosed in a simplex of $\mathcal{T}(R_s)$, solving point location in $\mathcal{A}(R_s^0)$ also identifies the cell of $\mathcal{T}(R_s)$ that contains the query (to ensure this, in preprocessing time we label each cell of $\mathcal{A}(R_s^0)$ by the cell of $\mathcal{T}(R_s)$ that encloses it). We know that $|R_s^0| \leq d^{O(d)} |R_s|^d$ (for a proof of this, see Lemma 7.2 in [3]). So $|R_s^0| = d^{O(d^2)}$.

Now it is clear that we use Meiser's algorithm to search in $\mathcal{A}(R_s^0)$ to locate the right simplex of $\mathcal{T}(R_s)$ that contains the query. The time needed to do one search in $\mathcal{A}(R_s^0)$ is $\tilde{O}(d^4) \log |R_s^0| = \tilde{O}(d^6)$. Since the depth of the main tree is $h = \lceil \log_{r_0} n \rceil$, the running time of the whole algorithm is $\tilde{O}(d^6)(\log n / \log r_0 + 1) = \tilde{O}(d^5)(\log n + d)$. The space is proportional to $|C_h|$ and hence it is $O(n^d)$. Since each simplex in the main tree uses a copy of Meiser's data structure for $|R_s^0|$ hyperplanes, the factor hidden by the big- O in space is $|R_s^0|^{d+\epsilon} = d^{O(d^3)}$.

Acknowledgments

We wish to thank Bernard Chazelle for reading a draft of this paper and giving several helpful comments and suggestions.

References

- [1] Chazelle, B. *Cutting hyperplanes for divide-and-conquer*, Discrete and Computational Geometry 9 (1993), 145–158.
- [2] Clarkson, K. L. *A probabilistic algorithm for the post office problem*, Proceedings 17th Annual ACM Symposium on Theory of Computing (STOC 1985), 175–184.
- [3] Clarkson, K. L. *New applications of random sampling in computational geometry*, Discrete and Computational Geometry 2 (1987), 195–222.
- [4] Matoušek, J. *Cutting hyperplane arrangements*, Discrete and Computational Geometry 6 (1991), 385–406.
- [5] Meiser, S. *Point location in arrangements of hyperplanes*, Information and Control 106 (1993), 286–303.