Efficient Answering of Polyhedral Queries in $\mathbb{R}^d$ Using BBS-Trees

K. Elbassioni * A. Elmasry † I. Kamel ‡

Abstract

We present a simple method for answering d-dimensional polyhedral queries, based on partitioning the space into disjoint regions and using a Balanced Binary Search (BBS) tree to index the points in each region. By appropriately selecting the boundaries of each region, we can guarantee an average search time that almost matches a known lower bound for the problem. Specifically, for a fixed d, if the coordinates of a given set of n points are statistically independent, the proposed technique answers polyhedral queries, on the average, in $O(mn^{1-1/d}.(\log n)^{1/d} + mk)$ using $O(n)$ space, where k is the number of reported points, and m is the number of linear constraints bounding the query region. Our approach is novel in that, while it provides a theoretical upper bound on the average query time, it avoids the use of complicated data structures, making it an effective candidate for practical applications.

1 Introduction

The problem of answering polyhedral queries has received reasonable attention. There are theoretically known lower and upper bounds for this problem. In particular, Chazelle and Rosenberg [3] show that simplex reporting in d-dimensions, using only linear $O(n)$ space, requires $\Omega(n^{1-1/d}+k)$ query time. For the upper bounds, Matoušek [5] gave an almost optimal algorithm for simplex range searching using *partition trees.* Given a static set of n points in d-dimensions and an $O(n)$ space, his technique can answer any simplex query in $O(n^{1-1/d+\epsilon} + k)$ time, after an $O(n \log n)$ preprocessing time, for any constant $\epsilon > 0$. However, this algorithm is not very practical since it uses complicated data structures, and the hidden constant in the query search time becomes large if a small ϵ is chosen. Some practical techniques have also been developed such as those using R-trees, R*-trees, etc [1, 4], but no theoretical bounds on their query time were shown.

In this paper, we present a method that lies in the middle between these two approaches. Under a natural assumption on the points′ coordinates, namely that they are statistically independent, we can answer a simplex query, on the average, in $O(n^{1-1/d}.(\log n)^{1/d} + k)$ using $O(n)$ space. This gives an upper bound that almost matches the worst-case bound obtained by the more complicated algorithms such as [5]. However, our algorithm is much simpler since it requires only BBS-trees for its implementation. Moreover, the query algorithm works directly on the original query region, and does not require partitioning it into simplices as is usually required by other algorithms.

It should also be mentioned that if space is not an issue and is allowed to increase non-linearly with n, then logarithmic query time can be achieved [2].

2 MB-index for answering polyhedral queries in $\mathbb{R}^d$

Given a set $\mathcal{P} \subseteq \mathbb{R}^d$ of n d-dimensional points distributed in a rectangular box $\mathcal{B} = [L_1, U_1] \times \ldots \times [L_d, U_d] \subseteq \mathbb{R}^d$, the location $x = (x_1, \ldots, x_d)$ of each point will be regarded as a random variable distributed in the interval $L_1 \le x_1 \le U_1, \ldots, L_d \le x_d \le U_d$, with probability density function $f(x_1, \ldots, x_d)$. The only assumption we make on the distribution of these locations is that they are statistically independent along each direction, i.e., $f(x_1, \ldots, x_d) = \prod_{i=1}^{d} f_i(x_i)$, where $f_i(x_i)$ is the probability density function along the ith direction.

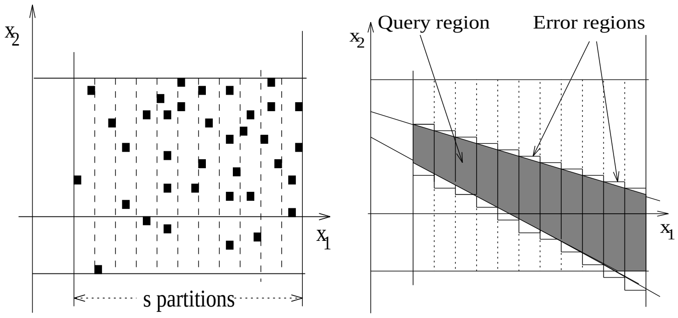
Given a set of half-spaces determined by the hyperplanes $\mathbb{H} = \{\mathcal{H}_1, \ldots, \mathcal{H}_m\}$, it is required to report all the points that lie in the intersection of these half-spaces. In this section, we develop an index structure, referred to as the *MB-index*, which enables us to efficiently answer such polyhedral queries about the given point set. More precisely, we shall prove the following result.

Theorem 1 *Under the statistical independence assumption, the average query time required by the MB-index to report all the points in the intersection of m half-spaces is $O(mn^{1-\frac{1}{d}} \log^{\frac{1}{d}} n + mk)$. The space required is $O(n)$, the preprocessing is $O(n \log n)$, and the amortized update time is $O(\log n)$.*

*Computer Science Department, Rutgers University, 110 Frelinghuysen Road, NJ 08854; (elbassio@paul.rutgers.edu).

†Computer Science Department, Rutgers University, 110 Frelinghuysen Road, NJ 08854; (elmasry@cs.rutgers.edu).

‡Panasonic Information and Networking Tech. Lab., Princeton, NJ 08540; (ibrahim@research.panasonic.com)



a: Partitioning the space. b: Partitioning the query region.

Figure 1: Partitioning the space and the query region.

Index structure and search algorithm. The general idea is to partition the space into disjoint rectangular regions, and index the points of each region on one of the dimensions using a BBS-tree (see Figure 1 for a two-dimensional example). Given a query polyhedron, the polyhedron will be intersected with each region, and the intersection will be approximated by a Minimum Bounding Box (MBB). Finally, the points in each MBB are reported including possibly some false hits (points reported in spite of the fact that they do not belong to the query region). By selecting the region boundaries so as to balance the number of points in each region, we can guarantee that the number of false hits is not too large.

In more details, to build the MB-index, we fix one dimension, say the dth dimension, and split the box $\mathcal{B}$ in each other dimension $i = 1, \ldots, d-1$, using s $(d-1)$-dimensional hyperplanes, perpendicular to the ith dimension, where s is to be determined later. Let $\mu_i(j)$ be the point on the ith axis defining jth hyperplane in the ith dimension (i.e., the hyperplane is $\{x \in \mathbb{R}^d \mid x_i = \mu_i(j)\}$), where $i = 1, \ldots, d-1$ and $j \in [s] \stackrel{\text{def}}{=} \{0, 1, \ldots, s-1\}$. Thus $\mu_i(0) = L_i$ and $\mu_i(s) = U_i$ for $i = 1, \ldots, d-1$. To bound the error probability, these hyperplanes will be chosen such that for all i and j

$$\int_{\mu_i(j)}^{\mu_i(j+1)} f_i(x_i)dx_i = \frac{1}{s} \tag{1}$$

is satisfied. In general, numerical (or analytical) integration can be used to obtain the required partitioning as described by (1), for any density function $f_i(x_i)$. Practically speaking, (1) says that the number of points in each interval should be $1/s$ of the total number n. Thus, for $i = 1, \ldots, d-1$ and $j \in [s]$, the region boundary $\mu_i(j)$ can be selected so as to make $|\{p \in \mathcal{P} \mid \mu_i(j) \le p_i \le \mu_i(j+1)\}| = n/s$.

Thus, with the above partitioning, we obtain a set of s^{d-1} sub-boxes $\{\mathcal{B}_J \mid J \in \mathcal{J}\}$, where $\mathcal{J} \stackrel{\text{def}}{=} \{(j_1, \ldots, j_{d-1}) \mid j_i \in [s], \text{ for all } i = 1, \ldots, d-1\}$, and $\mathcal{B}_J = \mathcal{B}_{j_1, \ldots, j_{d-1}}$ is the sub-box of $\mathcal{B}$ defined by the two corner points $(\mu_1(j_1), \ldots, \mu_{d-1}(j_{d-1}))$ and $(\mu_1(j_1+1), \ldots, \mu_{d-1}(j_{d-1}+1))$. A BBS-tree $\mathcal{T}(\mathcal{B}_J)$ is then constructed to index the points in each such sub-box $\mathcal{B}_J$. Obviously, the points in each of these BBS-trees are ordered by the value of their dth coordinate.

For a hyperplane $\mathcal{H} = \{x \in \mathbb{R}^d \mid a_1x_1 + \ldots + a_dx_d = c\}$, let us denote respectively by $\mathcal{H}^-$ and $\mathcal{H}^+$ the closed half-spaces $\{x \in \mathbb{R}^d \mid a_1x_1 + \ldots + a_dx_d \le c\}$ and $\{x \in \mathbb{R}^d \mid a_1x_1 + \ldots + a_dx_d \ge c\}$. Suppose we are interested in reporting all the points that lie on one side of $\mathcal{H}$, say $\mathcal{H}^-$. To do this, we intersect $\mathcal{H}$ with each of the sub-boxes $\mathcal{B}_J$. The highest point, with respect to the dth coordinate, in each intersection is determined, and all the points that lie below this point (have smaller value in the dth dimension) are reported. Next, each of these points is checked, and only accepted if it lies in the required half-space defined by $\mathcal{H}$. More precisely, for $i = 1, \ldots, d-1$, define

$$\alpha_i = \begin{cases} 1 & \text{if } a_i > 0 \\ 0 & \text{otherwise.} \end{cases} \tag{2}$$

Then the dth coordinates of the lowest and highest point in $\mathcal{B}_J$ that intersect $\mathcal{H}$, for $J = (j_1, \ldots, j_{d-1}) \in \mathcal{J}$, are given by

$$\lambda(J) = \frac{c}{a_d} - \sum_{i=1}^{d-1} \frac{a_i}{a_d} \cdot \mu_i(j_i + \alpha_i), \tag{3}$$

$$\Lambda(J) = \frac{c}{a_d} - \sum_{i=1}^{d-1} \frac{a_i}{a_d} \cdot \mu_i(j_i + 1 - \alpha_i), \tag{4}$$

provided $a_d \neq 0$. If $a_d = 0$, then $\lambda(J), \Lambda(J)$ are set to $\pm\infty$, depending on the signs of $c - \sum_{i=1}^{d-1} a_i \cdot \mu_i(j_i + \alpha_i)$, $c - \sum_{i=1}^{d-1} a_i \cdot \mu_i(j_i + 1 - \alpha_i)$, respectively.

Now, given a set $\mathbb{H} = \{\mathcal{H}_1, \ldots, \mathcal{H}_m\}$ of hyperplanes, where $\mathcal{H}_q = \{x \in \mathbb{R}^d \mid a_1^q x_1 + \ldots + a_d^q x_d = c^q\}$, let us assume, without loss of generality that, $a_d^q \ge 0$ for $q = 1, \ldots, m$. Suppose it is required to report all the points that lie in the intersection $\left(\bigcap_{q \in Q^+} \mathcal{H}_q^+\right) \bigcap \left(\bigcap_{q \in Q^-} \mathcal{H}_q^-\right)$, where $Q^+ \cup Q^- = \{1, \ldots, m\}$. In the following procedure, we denote by $\alpha_i^q, \lambda^q, \Lambda^q$, the values of these parameters as computed using (2), (3), and (4), with respect to the hyperplane $\mathcal{H}_q$.

Search Algorithm:
Input: An MB-index containing a set $\mathcal{P} \subseteq \mathbb{R}^d$ of n points, a set of m hyperplanes $\mathbb{H}$, and a partition $Q^+ \cup Q^-$ of $\{1, \ldots, m\}$.
Output: All points $p \in \mathcal{P}$ that lie in $\left(\bigcap_{q \in Q^+} \mathcal{H}_q^+\right) \cap \left(\bigcap_{q \in Q^-} \mathcal{H}_q^-\right)$.

For each $J \in \mathcal{J}$
1. Compute $\lambda^q(J), \Lambda^q(J)$, for $q = 1, \ldots, m$ as described above.
2. Let
   $l \leftarrow \text{argmax}\{\lambda^q(J) \mid q \in Q^+\}^a$,
   $r \leftarrow \text{argmin}\{\Lambda^q(J) \mid q \in Q^-\}$.
3. Search the BBS-tree $\mathcal{T}(\mathcal{B}_J)$ for the set of candidates
   $\mathcal{C} = \{p \in \mathcal{T}(\mathcal{B}_J) \mid \lambda^l(J) \leq p_d \leq \Lambda^r(J)\}$
   of the sub-box $\mathcal{B}_J$ satisfying the query.
4. For each $p \in \mathcal{C}$, if p lies inside the query polyhedron, report p.

[a] where argmax$\{\cdot\}$ means an index at which the maximum of $\{\cdot\}$ is attained

It is easy to see that the above procedure is conservative in the sense that no false dismissals are possible (all the points in the query region are reported). On the other hand, using the assumption about statistical independence and our partitioning strategy (1), we get the following key lemma, about the number of false hits, whose proof is given in the next section.

Lemma 1 *For any half-space query, the expected number of false hits is at most* $(d-1)\frac{n}{s}$.

Using this lemma, we can now prove Theorem 1:

Proof of Theorem 1. Let us first consider the query time. The total number of node accesses needed to search the s^{d-1} BBS-trees is bounded by $s^{d-1} \log n$. By Lemma 1, the expected number of false hits resulting from a single half-space query is at most $(d-1).n/s$. Hence, the expected total number of extra node accesses required to filter out the undesired points is at most $m(d-1).n/s$. The bound on the query time then follows by selecting

$$s = \left(\frac{n}{\log n}\right)^{\frac{1}{d}}.$$

As for the processing time, determining the boundaries $\mu_i(j)$ for $i = 1, \ldots, d-1$ and $j = 0, \ldots, s-1$ requires $O(s)$ if the density functions $f_i(x_i)$ are known. If these functions are not known, we partition each coordinate by counting the number of points, which can be done recursively in $O(n \log s)$. Next, determining the tree in which a point p should be stored requires finding, for each $i = 1, \ldots, d-1$, the minimum index j such that $\mu_i(j) \geq p_i$. This can be done in $O(\log s)$ by maintaining, for each i, a binary search tree on the boundaries $\mu_i(j)$. Thus we conclude that the preprocessing time required is $O(n \log n)$.

Finally, let us consider the time per update. Clearly, each insertion or deletion requires $O(\log n)$. However, after a number of updates, say insertions, the number of points in the different trees may increase significantly, possibly degrading the query performance. In that case, we select one (or more) over-crowded regions and split it into smaller regions. Since the split cost is non-trivial, it is reasonable to split only when the number of objects increases by some predefined factor, say 2. Similarly, if after deleting an object from a given tree, the number of objects drops below some factor times the original number, then the tree is merged with one or more adjacent trees. This way the split/merge operation will take $O(\log n)$ time in the amortized sense, and the theorem follows. □

3 Bounding the number of false hits - Proof of Lemma 1

To prove our key statement, Lemma 1, we shall need a few more definitions and two simple lemmas. For $i = 1, \ldots, d-1$, define $\mathcal{J}_i(0) = \{J \in \mathcal{J} \mid j_i = 0\}$, and $\mathcal{J}(0) = \cup_{i=1}^{d-1} \mathcal{J}_i(0)$. Given $J \in \mathcal{J}(0)$, we define *a chain* $\mathbb{C}(J)$ of points in $\mathcal{J}$ passing through J as follows:

$$\mathbb{C}(J) = \{I \in \mathcal{J} \mid I = J + k.(1-2\alpha),\ k \in \mathbb{Z}\},$$

where $\alpha = (\alpha_1, \ldots, \alpha_{d-1})$ is defined by (2). For $J \in \mathcal{J}(0)$, define $k_{min}(J) = \min\{k \in \mathbb{Z} \mid J + k.(1-2\alpha) \in \mathcal{J}\}$, and analogously define $k_{max}(J)$. Let us further index the elements of $\mathbb{C}(J)$ by their order in the chain, that is, let $\mathbb{C}(J) = \{J^k \mid k_{min}(J) \leq k \leq k_{max}(J)\}$, where $J^k = J + k.(1-2\alpha)$. Note that these definitions, and (3), (4) imply that $\lambda(J^{k+1}) = \Lambda(J^k)$ for $k_{min} \leq k \leq k_{max} - 1$.

Lemma 2 *The set of chains* $\{\mathbb{C}(J) \mid J \in \mathcal{J}(0)\}$ *partition the set of indices* $\mathcal{J}$. *That is,*

(i) *For all* $J, I \in \mathcal{J}(0)$, $\mathbb{C}(J) \cap \mathbb{C}(I) \neq \emptyset \Longrightarrow \mathbb{C}(J) = \mathbb{C}(I)$,

(ii) $\bigcup_{J \in \mathcal{J}(0)} \mathbb{C}(J) = \mathcal{J}$.

Proof. (i) Let $I, J \in \mathcal{J}(0)$ and suppose $F \in \mathbb{C}(I) \cap \mathbb{C}(J)$. Then there exist $k, k' \in \mathbb{Z}$ such that $F = J + k.(1-2\alpha)$ and $F = I + k'.(1-2\alpha)$, implying that $I = J + (k-k').(1-2\alpha)$. But then, for any $I' \in \mathbb{C}(I)$, there is a $k'' \in \mathbb{Z}$ such that $I' = I + k''.(1-2\alpha)$, and hence, $I' = J + (k'' + k - k').(1-2\alpha)$, i.e., $I' \in \mathbb{C}(J)$.

(ii) To show that chains cover all the elements of $\mathcal{J}$, consider an arbitrary element $J \in \mathcal{J}$. Clearly, letting $k = \min_{i=1}^{d-1}\{j_i \mid \alpha_i = 1\}$ (or $k = \max_{i=1}^{d-1}\{j_i \mid \alpha_i = -1\}$ if $\alpha_i = -1$ for all $i = 1, \ldots, d-1$), and letting $l \in \{1, \ldots, d-1\}$ be the index at which this minimum is attained, we get an $I \in \mathcal{J}_l(0)$ such that $J = I + k.(1-2\alpha)$, i.e., $J \in \mathbb{C}(I)$, for some $I \in \mathcal{J}(0)$. □

Lemma 3 *The total number of chains is at most* $|\mathcal{J}(0)| = s^{d-1} - (s-1)^{d-1} \leq (d-1).s^{d-2}$.

Proof of Lemma 1. Let $\mathcal{H} = \{x \in \mathbb{R}^d \mid a_1x_1 + \ldots + a_dx_d = c\}$ be the given hyperplane. (We may assume that $a_d \neq 0$. Otherwise, we can perturb $\mathcal{H}$ slightly by moving it a sufficiently small distance parallel to itself to guarantee that no points lie on its boundary, and then rotating it slightly, so that the query result does not change.) For each $J \in \mathcal{J}$, the probability of error in the sub-box $\mathcal{B}_J$ is strictly upper-bounded by the integration of $f(x_1, \ldots, x_d)$ over the part of $\mathcal{B}_J$ bounded in the dth dimension between $\lambda(J)$ and $\Lambda(J)$. Thus the total probability of error $\Pr[E]$ is upper-bounded by the sum of

$$\int_{\mu_1(j_1)}^{\mu_1(j_1+1)} \cdots \int_{\mu_{d-1}(j_{d-1})}^{\mu_{d-1}(j_{d-1}+1)} \int_{\lambda(J)}^{\Lambda(J)} f(x_1, \ldots, x_d)dx_1 \ldots dx_d.$$

over all $J = (j_1, \ldots, j_{d-1}) \in \mathcal{J}$.

Using independence and (1), the bound on the probability reduces to

$$\begin{aligned} \Pr[E] &\leq \sum_{J \in \mathcal{J}} \prod_{i=1}^{d-1} \int_{\mu_i(j_i)}^{\mu_i(j_i+1)} f_i(x_i)dx_i \int_{\lambda(J)}^{\Lambda(J)} f_d(x_d)dx_d \\ &= \frac{1}{s^{d-1}} \sum_{J \in \mathcal{J}} \int_{\lambda(J)}^{\Lambda(J)} f_d(x_d), \end{aligned}$$

and thus it is enough to show that

$$\epsilon \stackrel{\text{def}}{=} \sum_{J \in \mathcal{J}} \int_{\lambda(J)}^{\Lambda(J)} f_d(x_d)dx_d \leq (d-1)s^{d-2}. \qquad (5)$$

Now, we use Lemma 2 to decompose the error in (5) into the sum of errors over chains $\mathbb{C}(J)$, $J \in \mathcal{J}(0)$:

$$\begin{aligned} \epsilon &= \sum_{J \in \mathcal{J}(0)} \sum_{I \in \mathbb{C}(J)} \int_{x_d=\lambda(I)}^{\Lambda(I)} f_d(x_d)dx_d \\ &= \sum_{J \in \mathcal{J}(0)} \sum_{k=k_{min}(J)}^{k_{max}(J)} \int_{x_d=\lambda(J^k)}^{\lambda(J^{k+1})} f_d(x_d)dx_d \\ &= \sum_{J \in \mathcal{J}(0)} \int_{x_d=\lambda(J^{k_{min}(J)})}^{\Lambda(J^{k_{max}(J)})} f_d(x_d)dx_d. \end{aligned}$$

Since $\int_{x_d} f_d(x_d)dx_d \leq 1$, we conclude that $\epsilon \leq |\mathcal{J}(0)| \leq (d-1).s^{d-2}$ by Lemma 3. □

References

[1] N. Beckmann, H.-P. Kriegel, R. Schneider and B. Seeger, The R^*-tree: An Efficient and Robust Access Method for Points and Rectangles. In *Proc. ACM-SIGMOD International Conference on Management of Data*, 322–331, 1990.

[2] B. Chazelle, Cutting Hyperplanes for Divide-and-Conquer. *Discrete Computational Geometry*, 9:145–158, 1993.

[3] B. Chazelle and B. Rosenberg, Lower Bounds on the Complexity of Simplex Range Reporting on a Pointer Machine. In *Automata, Languages and Programming, 19th International Colloquium, ICALP92*, Vienna, Austria, July 1992, Lecture Notes in Computer Science, Vol. 623, 439–449, 1992.

[4] A. Guttman, R-trees: A Dynamic Index Structure for Spatial Searching. In *Proc. ACM-SIGMOD International Conference on Management of Data* , 47–57, 1984.

[5] J. Matoušek, Efficient Partition Trees. *Discrete Computational Geometry*, 8:432–448, 1992.