An Efficient Indexing Scheme for Multi-dimensional Moving Objects[⋆]

Khaled Elbassioni[1], Amr Elmasry[1], and Ibrahim Kamel[2]

[1] Computer Science Department, Alexandria University, Egypt
{elbassio,elmasry}@paul.rutgers.edu
[2] College of Information Systems, Zayed University, United Arab Emirates
Ibrahim.Kamel@zu.ac.ae

Abstract. We consider the problem of indexing a set of objects moving in d-dimensional space along linear trajectories. A simple disk-based indexing scheme is proposed to efficiently answer queries of the form: report all objects that will pass between two given points within a specified time interval. Our scheme is based on mapping the objects to a dual space, where queries about moving objects translate into polyhedral queries concerning their speeds and initial locations. We then present a simple method for answering such polyhedral queries, based on partitioning the space into disjoint regions and using a B-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 $O((N/B)^{1-1/d}.(\log_B N)^{1/d} + K/B)$ I/O's using $O(N/B)$ space, where B is the block size, and K is the number of reported points. 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

Maintaining a database of moving objects arises in a wide range of applications, including air-traffic control, digital battlefields, and mobile communication systems [3,7]. Traditionally, a database management system assumes that data stored in the database remain constant until they are explicitly modified through an update. With moving objects, storing the continuously changing locations of these objects directly in the database becomes infeasible, considering the large update overhead. An obvious solution, to overcome this problem, is to represent each object by its parameters (velocity and initial location), which will be stored in the database, and update the database only when one of these

⋆ The first and third authors gratefully acknowledge Panasonic Information and Networking Technologies Laboratory in Princeton, New Jersey, for its support during this work.

parameters changes. There has been some recent work on extending the current database technology to handle moving objects; see for example [18].

Given a database of N objects moving in d-dimensional space along linear trajectories, we consider, in this paper, the problem of constructing an index on these objects to efficiently answer range queries over their locations in the future. An example of such queries, in a database of moving cars, is: "Report all cars that will pass through some given region, within the next ten minutes". It is assumed that each object moves along a straight line; an assumption that applies to a large class of problems such as cars in an almost straight-line highway. Furthermore, many non-linear functions can be approximated by connected straight line segments.

Specifically, assume that the position of an object moving with velocity vector $v = (v_1, \ldots, v_d)$ starting from location $a = (a_1, \ldots, a_d)$, at time $t \geq 0$, is given by $y = a + vt$. Given two locations $y', y'' \in \mathbb{R}^d$, and two time instances t', t'' (where $y' \leq y''$ and $t' \leq t''$), it is required to report all objects which will pass between these two locations, during the time period $[t', t'']$ (see Figure 1:a for the one-dimensional case). In other words, we are interested in reporting all moving objects whose coordinates in the $(d+1)$-dimensional space $(t, y_1, \ldots, y_d)$ satisfy

$$\begin{array}{lcl} y'_i \leq & y_i(t) = v_i t + a_i & \leq y''_i, \text{ for } i = 1, \ldots, d, \\ t' \leq & t & \leq t'', \end{array} \tag{1}$$

where $y' = (y'_1, y'_2, \ldots, y'_d)$, $y'' = (y''_1, y''_2, \ldots, y''_d)$ and $y(t) = (y_1(t), \ldots, y_d(t))$.

In the standard external memory model of computation [4], the efficiency of an algorithm is measured in terms of the number of I/O's required to perform an operation. Let B be the page size, i.e., the number of units of data that can be processed in a single I/O operation. If K is the number of objects reported in a given query, then the minimum number of pages to store the data is $n \overset{\text{def}}{=} \lceil \frac{N}{B} \rceil$ and the minimum number of I/O's to report the answer is $k \overset{\text{def}}{=} \lceil \frac{K}{B} \rceil$. Thus the time and space complexity of a given algorithm, under such model, will be measured in terms of these parameters n and k. Finally to simplify the presentation, we shall assume, when using the $O(\cdot)$ notation, that the dimension d is constant.

The paper is organized as follows. In Section 2, we briefly survey related work. Section 3 states the main contribution of this paper and Section 4 gives an overview of the technique and the duality transformations used. We present the index structure and the search algorithm in Section 5, and the bound on the average query performance in Section 6. Preliminary experimental results on the one-dimensional case, comparing the performance of our method with other traditional techniques such as *R-trees*, are reported in Section 7. Finally, our conclusion is made in Section 8.

2 Related Work

One can generally distinguish two directions of research in the context of indexing moving objects. In the first one, techniques with theoretically guaranteed worst-case query time have been developed. Most of these techniques resort to *duality* to transform queries about moving objects to polyhedral queries involving their parameters (velocities and initial locations). For the latter problem, Matoušek [15] gave an almost optimal main memory 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 answers any simplex range query in $O(N^{(d-1)/d+\epsilon} + K)$ time, after an $O(N \log N)$ preprocessing time, for any constant $\epsilon > 0$. For the lower bound, Chazelle and Rosenberg [8] showed that simplex reporting in d-dimensions, using only linear space, requires $\Omega(N^{(d-1)/d} + K)$ I/O's. These bounds have been adapted to the external memory model by Kollios et al. [12], implying in particular, a query time of $O(n^{1-1/2d+\epsilon} + k)$ I/O's for answering queries of type (1), using linear space. Building on partition trees, Agarwal et al. [1] were able to obtain an index for moving objects in two-dimensions, for which the worst-case query time is $O(n^{\frac{1}{2}+\epsilon}+k)$ I/O's using $O(n)$ space. However, these techniques might be practically inefficient since they use complicated data structures, and the constants hidden under their complexity bounds might be quite large if a small ϵ is sought. 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; see [1,12] for examples. In the second approach, practical techniques have been developed that use the commercially available index structures such as B-trees, R-trees, R*-trees, etc [5,11,14]. Examples of this approach include the quad-tree method of [19], the time-parameterized R*-trees (*TPR trees*) of [16] and [6], and the work of Kollios et al. [12]. However, no theoretical bounds on the average query time were shown for these techniques.

In this paper, we propose a simple indexing structure that uses only B-trees in its implementation. We also provide an average case analysis of the query time given a random set of objects in the database. Our solution is based on a simple *internal memory* index structure, which we developed in [9], for answering polyhedral queries (i.e., queries determined by a finite set of linear constraints), whose average query time is as good as the worst-case query time obtained by the more complicated techniques. Such queries arise naturally in Spatial database applications; see [2,10] and the references therein.

3 The Contribution of the Paper

We can summarize the results of this paper as follows:

– Given a set of N points in $\mathbb{R}^d$, we propose an index structure to enable efficient answering of polyhedral queries about these points. Under a natural assumption on the points coordinates, namely that they are statistically independent, we can answer a query, on the average, in $O(mn^{1-1/d}.(\log_B n)^{1/d} + mk)$ I/O's using $O(n)$ space, where m is the number of linear constraints bounding the query region. Moreover, this result is valid for any data distribution (not necessarily uniform), and does not require the distribution function to be explicitly given. This gives an upper bound that almost matches the worst-case bound obtained by the more complicated algorithms such as [15]. However, our algorithm is much simpler since it requires only B-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.

– For a set of N objects moving in one-dimensional space, with any distribution of velocities and initial locations, we use the above method in the dual space to answer queries of type (1), on the average, in $O(\sqrt{n \log_B n} + k)$ I/O's using $O(n)$ space.
– For a set of N objects moving in d-dimensional space, with uniform distributions of velocities and initial locations, we answer queries (1), on the average, in $O(n^{1-1/3d}.(\log_B N)^{1/3d} + k)$ I/O's using $O(n)$ space.

4 Duality Transformations

One of the main challenges in indexing moving objects is that the trajectories of the objects are monotonically increasing with time. Thus, representing each object by a Minimum Bounding Box (MBB) is not appropriate since the overlap between the MBB's can be excessive, leading to bad performance. It is more efficient to represent each object by its parameters v_i and a_i and transform queries about moving objects into queries concerning these parameters. One commonly used transformation is to map each object with velocity vector $v = (v_1, \ldots, v_d)$ and initial location $a = (a_1, \ldots, a_d)$ into a $2d$-dimensional point $(v_1, a_1, \ldots, v_d, a_d)$ (for $d = 1$, this is called the Hough-X transform in [13]). Under this transformation, query region (1) becomes a $2d$-dimensional region, bounded by a set of linear constraints for $d = 1$, and a mix of linear and quadratic constraints for $d \geq 2$ (see Figure 1:b):

Proposition 1. *The d-dimensional query (1) can be expressed in the $(v_1, a_1, \ldots, v_d, a_d)$ space as follows:*

(i) *The d constraints* $\begin{cases} y_i' - v_i t'' \leq a_i \leq y_i'' - v_i t', & \text{if } v_i > 0 \\ y_i' - v_i t' \leq a_i \leq y_i'' - v_i t'', & \text{if } v_i \leq 0 \end{cases}$ *for $i = 1, \ldots, d$.*

(ii) *The set of $\binom{d}{2}$ constraints, for $1 \leq i < j \leq d$:*

$$\begin{cases} y_i'v_j - y_j''v_i \leq a_iv_j - a_jv_i \leq y_i''v_j - y_j'v_i, & \text{if } v_i > 0 \text{ and } v_j > 0 \\ y_i''v_j - y_j''v_i \leq a_iv_j - a_jv_i \leq y_i'v_j - y_j'v_i, & \text{if } v_i > 0 \text{ and } v_j \leq 0 \\ y_i'v_j - y_j'v_i \leq a_iv_j - a_jv_i \leq y_i''v_j - y_j''v_i, & \text{if } v_i \leq 0 \text{ and } v_j > 0 \\ y_i''v_j - y_j'v_i \leq a_iv_j - a_jv_i \leq y_i'v_j - y_j''v_i, & \text{if } v_i \leq 0 \text{ and } v_j \leq 0. \end{cases}$$

Proof. Use the Fourier-Motzkin elimination method [17] to eliminate the variable t from the set of constraints in (1). □

Since our proposed index can only deal with linear constraints, we shall get rid of such quadratic constraints by resorting to another type of duality transformation. Specifically, assuming $v_i \neq 0$ (for objects having $v_i = 0$, the problem can be reduced to a lower-dimensional problem; see Section 6), we let

$u_i \overset{\text{def}}{=} 1/v_i$ and $w_i \overset{\text{def}}{=} a_i/v_i$ be the new transformed parameters (this is the so-called Hough-Y transform in [13]). Note that for $v_i > 0$, this gives a one-to-one correspondence between (v_i, a_i) and (u_i, w_i). Thus, with such a transformation, our query region becomes a $2d$-dimensional polyhedron:

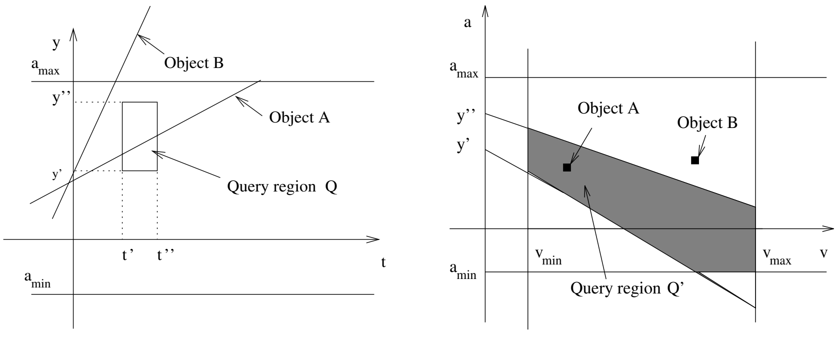
Proposition 2. *The d-dimensional query (1) can be expressed in the* $(u_1, w_1, \ldots, u_d, w_d)$*-space as follows:*

(i) *The d constraints* $\begin{cases} y'_i u_i - t'' \le w_i \le y''_i u_i - t', & \text{if } u_i > 0 \\ y''_i u_i - t'' \le w_i \le y'_i u_i - t', & \text{if } u_i \le 0 \end{cases}$ *for* $i = 1, \ldots, d$.

(ii) *The set of* $\binom{d}{2}$ *constraints, for* $1 \le i < j \le d$:

$$\begin{cases} y'_i u_i - y''_j u_j \le w_i - w_j \le y''_i u_i - y'_j u_j, & \text{if } u_i > 0 \text{ and } u_j > 0 \\ y'_i u_i - y'_j u_j \le w_i - w_j \le y''_i u_i - y''_j u_j, & \text{if } u_i > 0 \text{ and } u_j < 0 \\ y''_i u_i - y''_j u_j \le w_i - w_j \le y'_i u_i - y'_j u_j, & \text{if } u_i < 0 \text{ and } u_j > 0 \\ y''_i u_i - y'_j u_j \le w_i - w_j \le y'_i u_i - y''_j u_j, & \text{if } u_i < 0 \text{ and } u_j < 0 \ . \end{cases}$$

However, there is a problem with such transformation: the bound on the performance of our index uses the independence assumption. While it is natural to assume that the parameters $a_1, v_1, \ldots, a_d, v_d$ are statistically independent, this is not the case for the transformed variables $u_1, w_1, \ldots, u_d, w_d$. We handle this problem in Section 6: we use a property of the Hough-Y transform to show how the index structure can be modified in this case.



a: Native space. b: Dual space.

Fig. 1. Query region in the native and dual spaces for $d = 1$.

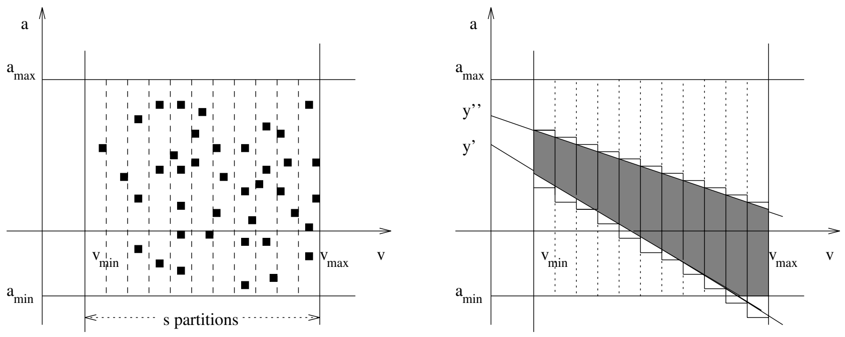
Having transformed the query region into a polyhedron, we develop, in the next section, an index structure, which we call *the MB-index*, to answer such queries efficiently. Thus, to summarize, for $d = 1$, we use the Hough-X duality transform and the MB-index in the 2-dimensional space (v_1, a_1). For $d \ge 2$, we use the Hough-Y transform and the MB-index in the $2d$-dimensional space $(u_1, w_1, \ldots, u_d, w_d)$.

5 MB-Index for Answering Polyhedral Queries in $\mathbb{R}^d$

Given a set $\mathcal{P}$ 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 can be regarded as a random variable distributed in the interval $L_1 \leq x_1 \leq U_1$, $\ldots$, $L_d \leq x_d \leq U_d$, with probability density function $f(x_1, \ldots, x_d)$. It is natural to assume that these locations 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. We further assume without loss of generality that the points are in general position.

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 describe an index structure which enables us to efficiently answer such polyhedral queries about the given point set.

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 B-tree. Given a query polyhedron, the polyhedron will be intersected with each region, and the intersection will be approximated by a Minimum Bounding Box (MBB) (see Figure 2). Finally, the points in each MBB are reported including possibly some false hits. 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.



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

Fig. 2. Partitioning the space and the query region.

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. To minimize the number of false hits (see Lemma 1 below), we select the number of partitions s as follows:

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

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 $j \in [s] \overset{\text{def}}{=} \{0, 1, \ldots, s-1\}$. 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

$$\int_{\mu_i(j)}^{\mu_i(j+1)} f_i(x_i)dx_i = \frac{1}{s} \tag{2}$$

is satisfied. Clearly, if the distributions of the coordinates are uniform, then the resulting partitions will be equi-distant. In general, numerical (or analytical) integration can be used to obtain the required partitioning as described by (2), for any density function $f_i(x_i)$. Practically speaking, (2) 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 $\mathcal{J} \overset{\text{def}}{=} \{(j_1, \ldots, j_{d-1}) \mid j_i \in [s],\ \text{for all } i = 1, \ldots, d-1\}$, 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 B-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 B-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

$$\alpha_i = \begin{cases} 1 & \text{if } a_i > 0 \\ 0 & \text{otherwise.} \end{cases} \tag{3}$$

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

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

provided $a_d \ne 0$. 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,

$a_d^q \geq 0$ for $r = 1, \dots, m$. 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 $Q^+ \cup Q^- = \{1, \dots, m\}$. In the following procedure, we denote by $\alpha_i^q, \lambda^q, \Lambda^q$, the values of these parameters as computed, using (3), (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 $\{1, \dots, m\}$.
Output: All points that lie in $\left(\bigcap_{q \in Q^+} \mathcal{H}_q^+\right) \bigcap \left(\bigcap_{q \in Q^-} \mathcal{H}_q^-\right)$.

```
For each $J \in \mathcal{J}$
    1. Compute $\lambda^q(J), \Lambda^q(J)$, for $q = 1, \dots, m$ 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 B-tree $\mathcal{T}(\mathcal{B}_J)$ for the set of candidates
           $\mathcal{C} = \{p \in \mathcal{T}(\mathcal{B}_J) \mid \lambda^l(J) \leq p_d \leq \Lambda^r(J)\}$
       inside 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 argmin means an index at which the minimum value is attained

It is easy to see that the above procedure is conservative in the sense that no false dismissals are possible. On the other hand, using the assumption about statistical independence and our partitioning strategy (2), the following key lemma, which is proved in [9], follows:

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

Insertions/Deletions. Finally, let us explain how to make our index dynamic. Clearly, each insertion/deletion requires $O(\log_B n)$ I/O's. However, after a number of updates, say insertions, the number of points in some trees may increase significantly, possibly degrading the query performance. In that case, every overcrowded tree is split into smaller trees. Since the split cost is non-trivial, it is reasonable to split only when the number of objects in the tree increases by some predefined factor. Similarly, if after deleting an object from a given tree, the number of objects drops below some factor of the original number, then the tree is merged with one or more adjacent trees. Moreover, when the total number of points increases/decreases by some predefined factor of the original number, the whole structure is rebuilt. This way the insert/delete operation will only take $O(\log_B n)$ I/O's in the amortized sense.

Using Lemma 1, we get the following result about the performance of the MB-index.

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

6 Bounding the Average Query Performance

As stated earlier, the bound on the MB-index performance is based on the independence assumption. However, this assumption can be violated if we use the transformation $u_i = \frac{1}{v_i}$, $w_i = \frac{a_i}{v_i}$. In this section, we discuss how to handle this problem.

Let us consider moving objects whose speeds and intercepts $v = v_i$ and $a = a_i$ in a given direction i are independent and uniformly distributed in the intervals $[v_{min}, v_{max}]$ and $[a_{min}, a_{max}]$ respectively. Assume without loss of generality that $a_{min} = -a_{max}$ (by translating the points along the a axis), and that $v_{min} > 0$ (points with $v_{min} < 0$ are handled similarly, while points with $v \approx 0$ are considered fixed and handled separately).

To be able to use the results of the previous section, we show in the Appendix that, the Hough-Y transform exhibits a nice property. Namely, for independent and uniformly distributed a, v, and for any $\eta' \leq \eta'', \mu' \leq \mu''$, the variables $u = 1/v$ and $w = a/v$ satisfy the inequalities

$$\int_{u=\eta'}^{\eta''} \int_{w=\mu'}^{\mu''} f(u,w)dudw \leq \begin{cases} 4\int_{\eta'}^{\eta''} g(u)du \int_{\mu'}^{\mu''} h(w)dw, & \text{if } \mu' \leq \frac{a_{max}}{\mathbb{E}[v]} \text{ and } \mu'' \geq \frac{a_{min}}{\mathbb{E}[v]} \\ \frac{1}{4}\left(1 - \frac{v_{min}}{v_{max}}\right)\int_{\eta'}^{\eta''} g(u)du, & \text{otherwise,} \end{cases} \tag{5}$$

where g and h are respectively the probability density functions of u and w, f their joint density function, and $\mathbb{E}[v] = (v_{max} + v_{min})/2$ is the average speed.

Then, using the MB-index technique of the previous section, we fix one direction, say u_d, and partition the space along each direction u_i, $i = 1, \ldots, d-1$ into s intervals determined by the points $\eta_i(0), \ldots, \eta_i(s)$, such that $\int_{\eta_i(j)}^{\eta_i(j+1)} g_i(u_i)du_i = 1/s$, where $s = (n/\log_B n)^{1/(2d)}$. Similarly, we partition the the space along each direction w_i, $i = 1, \ldots, d$ into s intervals determined by the points $\mu_i(0), \ldots, \mu_i(s)$, such that $\int_{\mu_i(j)}^{\mu_i(j+1)} h_i(w_i)dw_i = 1/s$. If for every $i = 1, \ldots, d$ we further have $v_{i,min}/v_{i,max} \geq 1 - 1/s$, then (5), applied to u_i, w_i for $i = 1, \ldots, d$, would imply that the probability of a false hit in any sub-region is upper-bounded as follows:

$$\begin{aligned} \Pr[FH] &= \int_{\eta_1(j_1)}^{\eta_1(j_1+1)} \int_{\mu_1(j_1')}^{\mu_1(j_1'+1)} \cdots \int_{\eta_d(j_d)}^{\eta_d(j_d+1)} \int_{\mu_d(j_d')}^{\mu_d(j_d'+1)} f(u_1, w_1, \ldots, u_d, w_d)du_1dw_1 \ldots du_d dw_d \\ &\leq 4^d \prod_{i=1}^{d} \int_{\eta_i(j_i)}^{\eta_i(j_i+1)} g_i(u_i)du_i \prod_{i=1}^{d} \int_{\mu_i(j_i')}^{\mu_i(j_i'+1)} h_i(w_i)dw_i. \end{aligned}$$

Consequently, we conclude by summing up all these probabilities that the total probability of error is upper-bounded by $\frac{4^d}{s}$ (see the proof of Lemma 1 in [9]). However, for $i = 1, \ldots, d$, the interval $[v_{i,min}, v_{i,max}]$ may be large making the assumption $v_{i,min}/v_{i,max} \geq 1 - 1/s$ invalid. To solve this problem, we partition this interval into $k = \lceil s \ln(v_{i,max}/v_{i,min}) \rceil$ intervals $[v_i(0), v_i(1)], \ldots, [v_i(k-1), v_i(k)]$, where $v_i(0) = v_{i,min}$, $v_i(k) = v_{i,max}$, and $v_i(j+1) = v_i(j).s/(s-1)$ for $j = 0, \ldots, k-1$. Clearly, in each such interval, the variable v_i is still uniformly distributed and $v_i(j)/v_i(j+1) \geq 1 - 1/s$, as required. Performing such a a partitioning for $i = 1, \ldots, d$, we obtain at most σs^d different regions, where $\sigma \overset{\text{def}}{=} \prod_{i=1}^{d} \ln(v_{i,max}/v_{i,min})$.

Thus to summarize, given a set of N d-dimensional moving objects, the index construction goes as follows:
1. Partition the set of points into 3^d groups according to whether $v_i < 0$, $v_i = 0$, or $v_i > 0$, for $i = 1, \ldots, d$.
2. Within each group, further partition the points, as explained above, into σs^d groups, according to which region of $[v_i(0), v_i(1)], [v_i(1), v_i(2)], \ldots$ contains v_i for $i = 1, \ldots, d$.
3. Finally, for each of the resulting groups, represent the points in the $(u_1, w_1, \ldots, u_d, w_d)$ space, and use an MB-index (consisting of s^{2d-1} B-trees) to store them.

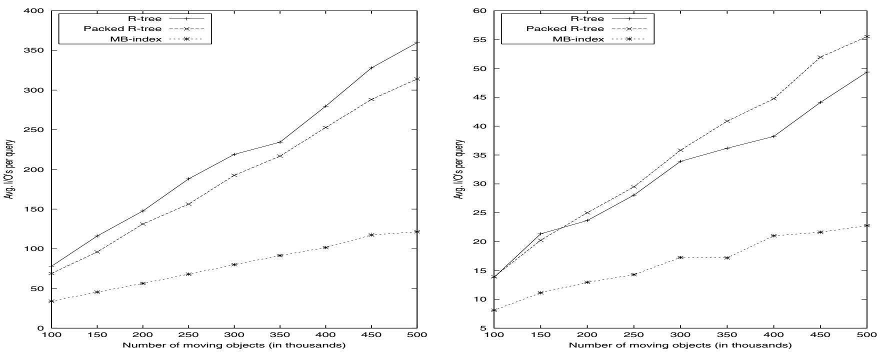
Let us now estimate the average query time of the resulting index. The total number of B-trees used is $\sigma(3s)^d.s^{2d-1} = \sigma 3^d s^{3d-1}$. The probability of error in each tree is at most $4^d/s$ as pointed out above. Thus selecting $s = \left(\frac{n}{\sigma \log_B n}\right)^{\frac{1}{3d}}$, we achieve an average search time of $O(n^{1-\frac{1}{3d}}(\sigma \log_B n)^{\frac{1}{3d}} + k)$, for a fixed d. Of course, for $d = 1$, we can use the MB-index directly in the (v_1, a_1)-space, and achieve an average query time of $O(\sqrt{n \log_B n} + k)$. We summarize our results in the following Theorem.

Theorem 2. *(i) For a set of N objects moving in one-dimensional space, with statistically independent velocities and initial locations, we can use the MB-index method to answer queries (1), on the average, in $O(\sqrt{n \log_B n} + k)$ I/O's using $O(n)$ space. The preprocessing is $O(N \log_B n)$, and the amortized update time is $O(\log_B n)$.*
(ii) For a set of N objects moving in d-dimensional space, with uniformly distributed and independent velocities and initial locations, queries (1) can be answered, on the average, in $O(n^{1-1/3d}(\sigma \log_B n)^{1/3d} + k)$ I/O's using $O(n)$ space. The preprocessing is $O(N \log_B n)$, and the amortized update time is $O(\log_B n)$.

7 Experimental Results

In this section, preliminary experimental results on the proposed indexing approach for the one-dimensional case are presented, and compared against R-tree based methods. As stated earlier, representing each object by a minimum bounding box (MBB) and using MBB-based indexing structures, such as R-trees, is not appropriate since the overlap between the MBB's will be excessive. Thus, to have a fair comparison with R-tree techniques, we first mapped each object into the dual space, where the query region becomes a trapezoid. Then we used the algorithm proposed in [10] to perform the intersection test between a trapezoid and a rectangle. For comparison purposes, two variants of R-trees have been used. The first is Guttman's R-tree [11] with quadratic splitting strategy. Objects were inserted incrementally, one at a time, so no preprocessing of the data was done. In the second variant, we used a *Packed R-tree*, where data is packed in the leaves of the tree in such a way to improve both node utilization and query processing time [14].

In the experiments, we used both uniform and non-uniform distribution of objects. In each case, two query sizes were used: 8% and 1% of the total space. The average number of I/O's over 1000 uniformly generated queries were then computed for each indexing technique. The page size used is 4096 bytes.



a: Query size=8% of space. b: Query size=1% of space.

Fig. 3. Average query performance for various indexing techniques with Uniform distribution.

In the first set of experiments, we generated $N = 100K, 200K, \ldots, 500K$ uniformly distributed objects, where each object was generated by picking, at random, its velocity v and its starting location a, where $v \in [0.16, 1.66]$, $a \in [0, 200]$. All speeds were assumed positive for simplicity. Figures 3:a,b present our results for the two types of queries. In these two figures, we show the average number of I/O's per query versus the number of objects in the database. As can be seen from the figures, the MB-index approach outperformed the R-tree based techniques. This gain in performance (approximately a factor of 2.5) is contributed to by two main components. The first is that the MB-index minimizes the dead space, so the number of I/O's can be significantly smaller. The second component is the fan-out of tree nodes. For the same page size, the B-tree has a larger fan-out than the R-tree since the B-tree is built on one dimensional data, while each entry in the R-tree contains a rectangle.

In the second experiment, we examined the effect of changing the distribution of speeds on the average number of I/O's for various techniques. For this experiment a normal distribution of speeds with mean $(v_{min} + v_{max})/2$ and standard deviation 1 was used. Table 1 presents the results for 8% and 1% queries. In the table, we show the average number of I/O's for different values of N. As seen from the table, while the performance of the simple R-tree degrades significantly with such distribution, both the packed R-tree and the MB-index methods remain almost invariant. The MB-index method remains superior to the packed R-tree technique for this case too.

Table 1. Average query performance for Normal distribution.

N	I/O's for query size=8%			I/O's for query size=1%		
	MB-index	R-tree	Packed R-tree	MB-index	R-tree	Packed R-tree
100K	31.908	489.998	68.281	8.553	244.096	14.634
150K	43.964	729.965	98.476	13.016	372.229	18.845
200K	56.232	966.029	128.455	13.870	493.500	24.282
250K	68.163	1209.458	157.413	14.926	624.128	29.860
300K	83.375	1441.948	190.943	18.961	740.153	33.986
350K	92.011	1688.847	217.671	21.987	868.695	39.315
400K	100.383	1933.160	252.462	22.858	990.933	45.241
450K	118.355	2146.008	280.440	22.529	1110.461	49.073
500K	140.457	2394.094	306.693	26.791	1245.052	54.106

8 Conclusion

In this paper we proposed a new technique for indexing objects moving in d-dimensional space along straight lines. We showed that this technique exhibits an efficient average query time under moderate assumptions on the object distributions. A by-product of our technique is an efficient method for indexing multi-dimensional queries determined by linear constraints. One distinguishing feature of our index is its simplicity which makes it practically applicable. Experimental results indicate that this technique outperforms, by a factor of almost 2.5, the performance of other traditional methods based on R-trees. In future work, we intend to address extensions for other types of queries.

References

1. P. K. Agarwal, L. Arge and J. Erickson, Indexing Moving points. In *Proc. 19th ACM PODS*, pp. 175–186, 2000.
2. P. K. Agarwal, L. Arge, J. Erickson, P. Franciosa and J. S. Vitter, Efficient Searching with Linear Constraints. In *Proc. 17th ACM PODS* , pp. 169–178, 1998.
3. R. Alonso and H. F. Korth, Database System Issues in Nomadic Computing. In *Proc. ACM-SIGMOD International Conference on Management of Data* , pp. 388–392, 1993.
4. A. Aggarwal and J. S. Vitter, The Input/Output Complexity of Sorting and Related Problems. In *Communications of the ACM*, 31(9):1116–1127, 1988.
5. 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*, pp. 322–331, 1990.
6. M. Cai, D. Keshwani and P. Z. Revesz, Parametric rectangles: A model for querying and animating spatiotemporal databases. In *Proc. 7th International Conference on Extending Database Technology*, LNCS 1777, pp. 430–444. Springer, 2000.
7. S. Chamberlain, Model-based battle command: A paradigm whose time has come. In *Proc. 1st International Symposium on Command and Control Research and Technology*, pp. 31–38, 1995.

8. B. Chazelle and B. Rosenberg, Lower Bounds on the Complexity of Simplex Range Reporting on a Pointer Machine. In *Proc. 19th International Colloquium on Automata, Languages and Programming*, LNCS, Vol. 693, 1992.
9. K. Elbassioni, A. Elmasry and I. Kamel, Efficient Answering of Polyhedral Queries in $\mathbb{R}^d$ using BBS-trees. In *Proc. 14th Canadian Conference on Computational Geometry (CCCG 2002)*, pp. 54–57, 2002.
10. J. Goldstein, R. Ramakrishnan, U. Shaft and J. B. Yu, Processing Queries by Linear Constraints. In *Proc. 16th ACM PODS*, pp. 257–267, 1997.
11. A. Guttman, R-trees: A Dynamic Index Structure for Spatial Searching. In *Proc. ACM-SIGMOD*, pp. 47–57, 1984.
12. G. Kollios, D. Gunopulos and V. Tsotras, On Indexing Mobile Objects. In *Proc. 18th ACM PODS,* pp. 261–272, 1999.
13. H. V. Jagadish, On Indexing Line Segments. In *Proc. 16th VLDB Conference* , pp. 614–625, 1990.
14. I. Kamel and C. Faloutsos, On Packing R-trees. In *Proc. Second International Conference on Information and Knowledge Management*, 1993.
15. J. Matoušek, Efficient Partition Trees. *Disc. and Computational Geometry*, 8 (1992), pp. 432–448.
16. S. Šaltenis, C. S. Jensen, S. T. Leutenegger, and M. A. Lopez, Indexing the Positions of Continuously Moving Objects. In *Proc. ACM-SIGMOD*, pp. 331–342, 2000.
17. A. Schrijver. *Theory of Linear and Integer Programming*, Wiley-Interscience, 1986.
18. A. P. Sistla, O. Wolfson, S. Chamberlain and S. Dao, Modeling and Querying Moving Objects. In *Proc. 13th IEEE ICDE Conference*, pp. 422–432, April 1997.
19. J. Tayeb, O. Ulusoy and O. Wolfson, A quadtree-Based Dynamic Attribute Indexing Method. *The Computer Journal*, 41(3):185–200, 1998.

Appendix: Proof of Inequality (5)

Define $L = \int_{u=\eta'}^{\eta''} \int_{w=\mu'}^{\mu''} f(u,w)dudw$ and $R = \int_{u=\eta'}^{\eta''} g(u)du \int_{w=\mu'}^{\mu''} h(w)dw$, and note that

$$L = \Pr[\frac{1}{\eta''} \le v \le \frac{1}{\eta'}, \mu' v \le a \le \mu'' v],$$
$$R = \Pr[\frac{1}{\eta''} \le v \le \frac{1}{\eta'}]. \Pr[\mu' v \le a \le \mu'' v].$$

By uniformity and independence of v, a, these probabilities are proportional to the areas of the corresponding regions in the (v, a) space (in particular, L is equal to the area of the shaded region in Figure 4). We consider 7 cases according to how the lines $a = \mu' v$ and $a = \mu'' v$ cross the borders of the probability region (Figures 4:a,b, and 8:c-g, respectively). Any other case is implied by, or can be reduced to, one of these cases. Below, we use for brevity the notation $h_t = v_{max} - v_{min}$ and $a_t = a_{max} - a_{min}$. The symbols $x, x', y, y', y'', h', h''$, and h''', in each case, denote the distances shown in the corresponding figure.

Case 1. $L = \frac{(x+y)h}{2a_t h_t}$, $R = \frac{(x'+y')h_t}{2a_t h_t} \cdot \frac{h}{h_t}$. Then $\frac{L}{R} = \frac{x+y}{x'+y'} < 2$. (see Figure 4:a.)

Case 2. $\frac{L}{R} = \frac{2x'}{2x'h'/h_t+(x'+y)(h''-h')/h_t+(y+y')(1-h''/h_t)}$. Since $\frac{y}{x'} \geq 1 - \frac{h''}{h_t}$, we get $\frac{L}{R} \leq \frac{2x'}{x'(h'+h'')/h_t+x'(1-h'/h_t)(1-h''/h_t)+y'(1-h''/h_t)} < 2$.

Case 3. $\frac{L}{R} = \frac{2x'h'/h+(x'+x)(h''-h')/h+(x+y)(h-h'')/h}{2x'h'/h_t+(x'+x)(h''-h')/h_t+(x+y')(1-h''/h_t)} < 2$, since $\frac{x}{x'} \geq 1 - \frac{h''}{h_t}$.

For cases 4,5, and 6 below, assume first that $\mu'' \geq \frac{a_{min}}{\mathbb{E}[v]}$.

Case 4. $\frac{L}{R} = \frac{x'+x}{yh'/h_t+y'(h'-h'')/h_t}$. Using the assumption $\mu'' \geq \frac{a_{min}}{\mathbb{E}[v]}$, we get $h'/h_t \geq 1/2$, implying that $\frac{L}{R} \leq \frac{2y}{y/2+y'(h'-h'')/h_t} < 4$.

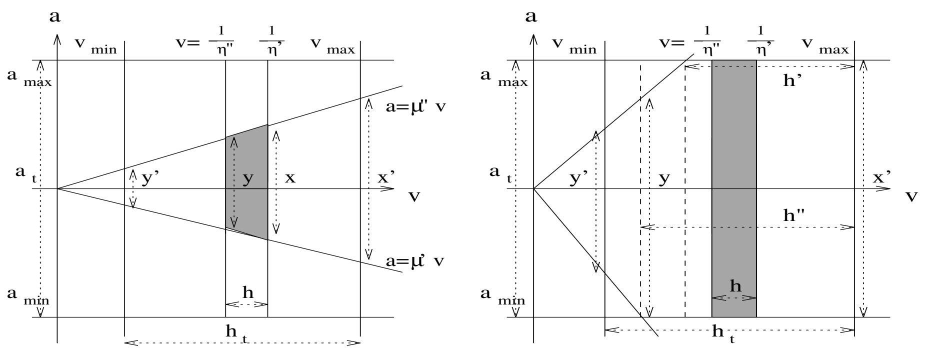
Case 5. $\frac{L}{R} = \frac{(x+y)}{x'h''/h_t+(x'+y')(h'-h'')/h_t} \leq \frac{2x'}{x'h'/h_t+y'(h'-h'')/h_t} < 4$.

Case 6. $\frac{L}{R} = \frac{(x'+x)(h''-h''')/h+(x+y)(h'''+h-h'')/h}{xh''/h_t+(x+y')(h'-h'')/h_t} \leq \frac{2x}{xh'/h_t+y'(h'-h'')/h_t} < 4$.

On the other hand, if $\mu'' < \frac{a_{min}}{\mathbb{E}[v]}$, that is, if $h'/h_t < 1/2$, then as we can see in Figure 4:g, the probability L is bounded by h/h' times the ratio of the area of the shaded region to the total area of the probability space. Noting that $a_{min} = -a_{max}$ and $h/h_t = \int_{u=\eta'}^{\eta''} g(u)du$, we conclude that

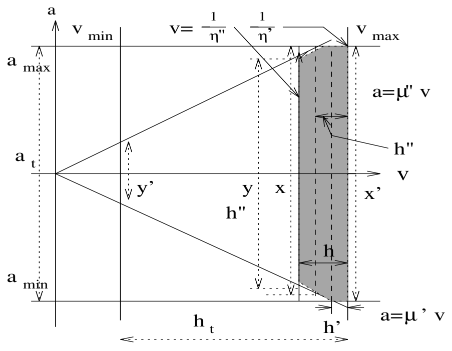
$$L \leq \frac{(\mu'' v_{min} - a_{min})h}{2a_t h_t} < \frac{1}{4}\left(1 - \frac{v_{min}}{\mathbb{E}[v]}\right) \cdot \frac{h}{h_t} < \frac{1}{4}\left(1 - \frac{v_{min}}{v_{max}}\right)\int_{u=\eta'}^{\eta''} g(u)du$$

as desired. $\square$

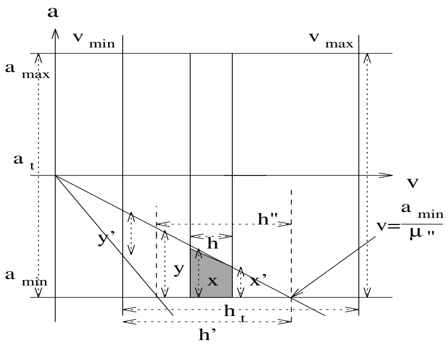


a: Case 1. b: Case 2.

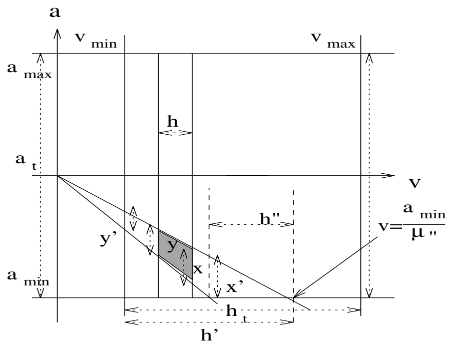
Fig. 4. The case analysis proof of Inequality (5).



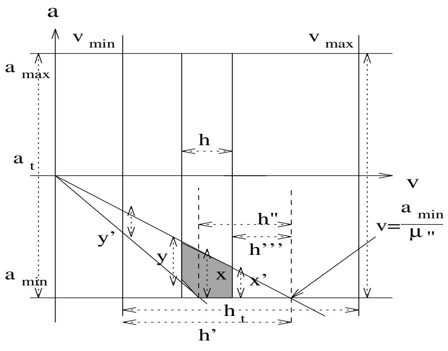
c: Case 3.



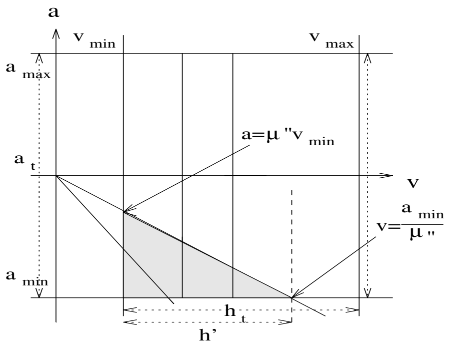
d: Case 4.



e: Case 5.



f: Case 6.



g: Case $\mu'' < \frac{a_{min}}{\mathbb{E}[v]}$.

Fig. 4 (Cont.) The case analysis proof of Inequality (5).