

# On Nearest Neighbor Indexing of Nonlinear Trajectories

Charu C. Aggarwal  
IBM T. J. Watson Research Center  
19 Skyline Drive  
Hawthorne, NY 10532  
charu@us.ibm.com

Dakshi Agrawal  
IBM T. J. Watson Research Center  
19 Skyline Drive  
Hawthorne, NY 10532  
agrawal@us.ibm.com

## ABSTRACT

In recent years, the problem of indexing mobile objects has assumed great importance because of its relevance to a wide variety of applications. Most previous results in this area have proposed indexing schemes for objects with linear trajectories in one or two dimensions. In this paper, we present methods for indexing objects with nonlinear trajectories. Specifically, we identify a useful condition called the *convex hull property* and show that any trajectory satisfying this condition can be indexed by storing a careful representation of these objects in a traditional index structure. Since a wide variety of relevant nonlinear trajectories satisfy this condition, our result significantly expands the class of trajectories for which nearest neighbor indexing schemes can be devised. We also show that even though many non-linear trajectories do not satisfy the convex hull condition, an approximate representation can often be found which satisfies it. We discuss examples of techniques which can be utilized to find representations that satisfy the convex hull property. We present empirical results to demonstrate the effectiveness of our indexing method.

## 1. INTRODUCTION

The problem of indexing mobile objects has recently attracted the interest of the database community because of an increasing number of applications involving spatio-temporal data [1, 8, 12, 15]. In many applications such as radar tracking and geographical information system (GIS), the objects in the database are entities whose location may vary with time. Specifically, the trajectory of each object is defined as a function of time  $f(t)$ . An application may track a large number of such objects in a database, and it is often desirable to query the database at specific instances in time with questions such as:

- At a given instant in time  $t$ , find the  $k$  closest objects to a given location  $q$  (Nearest Neighbor Query).
- At a given instant in time  $t$ , find the  $k$  closest objects

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

PODS 2003, June 9-12, 2003, San Diego, CA  
Copyright 2003 ACM 1-58113-670-6/03/06 ...\$5.00.

to a given line  $l$  (Linear Optimization Query).

- At a given instant in time  $t$ , find all objects within a user specified range  $R$  (Range Query).

We note that traditional database indexing structures [2, 3, 5, 6, 10, 17] cannot be used for such continuously changing data, since we would need to update the database at each instant in time. In recent years, a number of indexing and query processing techniques have been proposed for the problem of indexing mobile objects [1, 4, 7, 8, 11, 12, 13, 15, 16, 18, 19, 20]. Most of these techniques assume that the objects have linear trajectories in one or two dimensions. In many real applications, these may turn out to be rather restrictive assumptions. For example, radar tracking applications are inherently three dimensional. Similarly many trajectories, such as those of objects moving under the influence of gravity may be inherently nonlinear.

In general, the wide variation in the nature of nonlinear trajectories makes the indexing problem quite difficult. In many practical applications, an approximate solution can be provided by using piecewise linear approximations of nonlinear trajectories. In this paper, we will provide an indexing scheme which can be utilized for indexing a large class of nonlinear trajectories in arbitrary dimensions. We will identify the class of trajectories amenable for indexing by a simple property referred to here as the *convex hull property*. We will show that many important classes of trajectories can be either exactly or approximately represented in this form. In order to illustrate the proposed technique, this paper primarily analyzes the nearest-neighbor query problem, but the results can be easily extended to queries of other types. We will provide a brief description and justification of how the nearest neighbor method discussed in this paper can also be extended to the case of other kinds of queries such as linear optimization queries.

This paper is organized as follows. In the next section, we will discuss the convex hull property which describes the class of indexable trajectories. In Section 3, we will discuss the indexing scheme and query processing of objects whose trajectories satisfy the convex hull property. In Section 4, the empirical results are discussed. Finally, Section 5 contains the conclusion and summary.

### 1.1 Contributions of this paper

This paper discusses the first indexing scheme for objects with nonlinear trajectories in arbitrary dimensions. Previous results in this area are only applicable to indexing objects with linear trajectories in one or two dimensions. This paper significantly extends the range and scope of indexable

mobile objects. Specifically, we define a convex hull property and show that the technique proposed in this paper can handle all objects whose trajectories satisfy this property. We show that many important classes of nonlinear trajectories can be either exactly or approximately represented in a form which satisfies the convex hull property. We also discuss ways in which such approximate representations could be found. In addition, we show that the results of this paper are also applicable to other kinds of queries such as linear optimization queries.

## 2. FUNCTIONAL REPRESENTATIONS AND THE CONVEX HULL PROPERTY

In order to develop the ideas in this paper further, we will first introduce some notations and definitions. We assume that the database has  $N$  mobile objects, each of which has a trajectory with dimensionality  $d$ . We assume that the position of the  $i$ -th object at time  $t$  is given by a function  $F(\Theta^i = (\theta_1^i, \dots, \theta_k^i), t)$  of  $k$  parameters. We refer to  $F(\Theta, t)$  as the *functional representation* of the trajectory and to  $\Theta$  as the *parametric representation* associated with the functional representation  $F(\cdot, t)$ . It may happen that the trajectory  $F(\Theta, t)$  changes at particular instants of time, in which case  $(\Theta)$  may need to be updated. The *instantiation* of the functional representation  $F(\Theta, t)$  at time  $t = t_0$  is the *snapshot* of the object at time  $t_0$ . The following example clarifies these definitions:

**EXAMPLE 2.1.** *The trajectory (height) of a projectile thrown up vertically from height  $y$  at velocity  $v$  is given by:*

$$F(y, v, -g, t) = y + v \cdot t + (-g) \cdot t^2$$

Here  $g$  is the gravitational acceleration. One functional representation of the trajectory is given by  $F_1(\Theta = (\theta_1, \theta_2, \theta_3), t) = \theta_1 + \theta_2 t - \theta_3 t^2$ , where the parametric representation associated with this functional representation is  $(y, v, -g)$ . An alternative functional representation of the trajectory is  $F_2(\Theta = (\theta_1, \theta_2, \theta_3), t) = \theta_1 + \theta_2 t - (\theta_3 t)^2$ . The parametric representation associated with  $F_2(\cdot, t)$  is  $(y, v, \sqrt{g})$ .

As illustrated by the above example, a trajectory may admit many functional representations. The *convex hull property* is defined as a property of the functional representation and the associated parametric representation of a given trajectory.

**DEFINITION 2.1.** *A functional representation  $F(\Theta, t)$  and its associated parametric representation  $\Theta$  are said to satisfy the convex hull property when for any set of  $N$  trajectories with functional representations  $F(\Theta^1, t), \dots, F(\Theta^N, t)$  and the corresponding parametric representations  $\Theta^1 = (\theta_1^1, \dots, \theta_k^1), \dots, \Theta^N = (\theta_1^N, \dots, \theta_k^N)$  the following holds true: If  $\Theta'$  lies inside the convex hull of  $\Theta^1, \dots, \Theta^N$  then  $F(\Theta', t)$  lies inside the convex hull of  $F(\Theta_1^1, t), \dots, F(\Theta_1^N, t)$ .*

We note that the convex hull property is not a property of the trajectory itself, but it is a property of a particular parametric representation of it. It may often happen that the associated parametric representation for one functional representation may satisfy the convex hull property, whereas another may not. In Example 1, it turns out that while the first functional representation satisfies the convex hull property, the second does not. For ease of exposition, we

will loosely say that a trajectory satisfies the convex hull property if a functional representation could be found which satisfies the convex hull property. The following observation establishes the convex hull property for an important class of trajectories:

**OBSERVATION 2.1.** *A  $d$ -dimensional trajectory with constant velocity satisfies the convex hull property.*

A  $d$ -dimensional trajectory with constant velocity can be represented by parameters  $(\bar{y}, \bar{v})$ , where  $\bar{y}$  is the position of the object at time  $t = 0$  and  $\bar{v}$  is the constant velocity. The corresponding functional representation is given by  $F(\bar{y}, \bar{v}, t) = \bar{y} + \bar{v} \cdot t$ .

It is easy to see that the above observation is true. Consider  $N$  trajectories with parametric representations  $(\bar{y}_1, \bar{v}_1), (\bar{y}_2, \bar{v}_2), \dots, (\bar{y}_N, \bar{v}_N)$ . The corresponding functional representations at time  $t$  are given by  $\bar{y}_1 + \bar{v}_1 \cdot t, \bar{y}_2 + \bar{v}_2 \cdot t, \dots, \bar{y}_N + \bar{v}_N \cdot t$ . Consider a trajectory with parametric representation  $(\bar{y}', \bar{v}')$  which lies in the convex hull of  $(\bar{y}_1, \bar{v}_1), \dots, (\bar{y}_N, \bar{v}_N)$ . From the definition of convexity, this means that there exists a vector  $(\lambda_1, \dots, \lambda_N)$  which satisfies the following conditions:

$$(\bar{y}', \bar{v}') = \sum_{i=1}^N \lambda_i \cdot (\bar{y}_i, \bar{v}_i), \quad \sum_{i=1}^N \lambda_i = 1 \quad (1)$$

Now, by scaling and addition of the various components of the conditions discussed above, it is easy to see that the following must be true as well:

$$\bar{y}' + \bar{v}' \cdot t = \sum_{i=1}^N \lambda_i \cdot (\bar{y}_i + \bar{v}_i \cdot t), \quad \sum_{i=1}^N \lambda_i = 1 \quad (2)$$

This means that  $(\bar{y}' + \bar{v}' \cdot t)$  lies in the convex hull of  $(\bar{y}_1 + \bar{v}_1 \cdot t), \dots, (\bar{y}_N + \bar{v}_N \cdot t)$ . Therefore,  $d$ -dimensional trajectories with constant velocity satisfy the convex hull property. The above observation can be generalized as follows:

**OBSERVATION 2.2.** *The  $d$ -dimensional trajectory with polynomial snapshot representation at time  $t$  given by  $\sum_{i=0}^{k-1} \mathbf{a}_i \cdot t^i$ ,  $\mathbf{a}_i \in \mathcal{R}^d$ , satisfies the convex hull property.*

The method for proving this observation is exactly analogous to the above proof. We note that the above observation includes multi-dimensional parabolic trajectories. These trajectories are important because projectiles falling under the effect of gravity fall in this class. Another important observation is that a parabolic trajectory can be parameterized by either cartesian coordinates or by polar coordinates. While the cartesian representation satisfies the convex hull property, the polar representation does not. Therefore, we note that it may often be a matter of skill, experience, and appropriate approximation to pick a functional representation for a trajectory so that it satisfies the convex hull property. In particular, the observation discussed above for polynomial trajectories has significant implications for *approximate* representation of many general classes of trajectories.

This is because in many practical applications, even though the exact position of an object may vary with time, the range of time over which one needs to query is known in advance. In such cases, it is often possible to provide an approximate Taylor Expansion of the trajectory. This expansion

can often approximate arbitrary functional trajectories in a limited number of polynomial terms for restricted ranges of the function. The polynomial nature of the expansion lends immediate applicability of the convex hull property. The Taylor Expansion is defined as follows:

DEFINITION 2.2. *If a function  $f(x)$  has continuous derivatives up to the  $(n + 1)$ th order, then it can be expanded in the following fashion:*

$$f(x) = f(a) + f'(a) \cdot (x - a) + \dots + \frac{f^{(n)}(a) \cdot (x - a)^n}{n!} + R_n \quad (3)$$

where  $R_n$ , called the remainder after  $n + 1$  terms is given by:

$$R_n = \int_a^x f^{(n+1)}(u) \cdot \frac{(x - u)^n}{n!} du \quad (4)$$

When this expansion converges over a certain range of  $x$ , that is  $\lim_{n \rightarrow \infty} R_n = 0$ , then the expansion is called the Taylor Series of  $f(x)$  expanded about  $a$ .

We note that by using a careful choice of  $a$ , it is possible to approximate the function closely. In many cases, only a small number of terms of the expansion are required in order to closely approximate the function in restricted ranges. Examples of such functions include the exponential function, and the logarithmic function among others. A detailed description of the characteristics and convergence properties of the Taylor expansion may be found in [9].

Another example of an important trajectory which can be made to satisfy the convex hull property with a careful choice of parametric representation is a set of objects moving in the following elliptical orbits:

EXAMPLE 2.2. *Consider a large set of objects, such that the  $x$ -coordinate and  $y$ -coordinate of the  $i$ th object at time  $t$  are defined as follows:*

$$x = a_i \cdot \text{cosine}(t - t_i) \quad (5)$$

$$y = b_i \cdot \text{sine}(t - t_i) \quad (6)$$

We assume that  $a_i$ ,  $b_i$  and  $t_i$  are constants which are specific to the object  $i$ .

Thus, the  $i$ th object moves along the elliptical orbit defined by  $x^2/a_i^2 + y^2/b_i^2 = 1$ . The values of  $a_i$  and  $b_i$  are the minor and major axes of the ellipse, whereas the constant  $t_i$  determines the starting point of each of the objects at reference time  $t = 0$ . We note that the natural parametric representation  $(a_i, b_i, t_i)$  does *not* satisfy the convex hull property. However, it is possible to find a representation which satisfies the convex hull property by expanding the trajectory terms as follows:

$$x = a_i \cdot \text{cosine}(t) \cdot \text{cosine}(t_i) + a_i \cdot \text{sine}(t) \cdot \text{sine}(t_i) \quad (7)$$

$$y = b_i \cdot \text{sine}(t) \cdot \text{cosine}(t_i) - b_i \cdot \text{cosine}(t) \cdot \text{sine}(t_i) \quad (8)$$

Next, we redefine the following parameters:

$$c_i = a_i \cdot \text{cosine}(t_i) \quad (9)$$

$$d_i = a_i \cdot \text{sine}(t_i) \quad (10)$$

$$e_i = b_i \cdot \text{cosine}(t_i) \quad (11)$$

$$f_i = -b_i \cdot \text{sine}(t_i) \quad (12)$$

Now we note that the trajectory of object  $i$  can be expressed in terms of the parameters  $c_i$ ,  $d_i$ ,  $e_i$  and  $f_i$  as follows:

$$x = c_i \cdot \text{cosine}(t) + d_i \cdot \text{sine}(t) \quad (13)$$

$$y = e_i \cdot \text{sine}(t) + f_i \cdot \text{cosine}(t) \quad (14)$$

For each object  $i$ , we choose  $(c_i, d_i, e_i, f_i)$  as its parametric representation. It can be shown in an analogous way to the proof of convexity property of the linear trajectory, that the above representation also satisfies the convex hull property.

The above analysis holds for the case when the major and minor axis of each of the orbits is oriented along the  $X$ -axis and  $Y$ -axis. In many cases, the major and minor axes of the different orbits can be oriented differently. The functional representation for such objects is defined as follows:

$$x = a_i^1 \cdot \text{cosine}(t - t_i) + a_i^2 \cdot \text{sine}(t - t_i) \quad (15)$$

$$y = b_i^1 \cdot \text{sine}(t - t_i) + b_i^2 \cdot \text{cosine}(t - t_i) \quad (16)$$

The argument for the convexity of axis-parallel elliptical trajectories can be directly extended to such arbitrarily oriented trajectories by expansion of the corresponding terms  $\text{cosine}(t - t_i)$  and  $\text{sine}(t - t_i)$  in the above expressions. The constant coefficients of this expanded functional representation can be used as the indexed parameters.

In the next section, we will discuss how the carefully chosen parametric representations can be utilized in order to construct the index structures for effective nearest neighbor indexing.

### 3. INDEXING AND QUERY PROCESSING TECHNIQUES

The convex hull property relates the locality in parametric space to the locality in the positions of mobile objects. This fact can be exploited for efficient query processing of the locations of mobile objects by *indexing the parametric representations* of their trajectories. As an application of this technique, we will demonstrate its use to resolve the nearest neighbor query. In the following, we will first discuss the details of a branch and bound query processing technique on the index structure built on the parametric representations. Later, we will prove its correctness using the convex hull property.

Assume that an application tracks  $N$  objects for which the corresponding trajectories satisfy the convex hull property. For each object, we maintain a suitable parametric representation of its trajectory in a multi-dimensional index structure such as [2]. Thus, for example, consider the parabolic trajectory with parameters  $(y, v, a)$ , and snapshot representation at time  $t$  given by  $y + v \cdot t + a \cdot t^2$ . Then, the parameter values for the different objects are maintained in a multidimensional index tree structure. This index structure is updated whenever the trajectory of an object changes.

The overall branch and bound technique for the nearest neighbor query is illustrated in Figure 1. The input to the algorithm is the query point  $q$ , the time  $t$  at which the nearest object is to be found, and the root node  $RN$  of the index structure. The branch and bound technique utilizes a hierarchical traversal of the tree structure in order to visit the nodes which are most closely related to the query point. At each stage of the algorithm, we maintain a set of nodes, denoted by LIST, which will be explored further by the algorithm. The value of LIST is initialized to the root node  $RN$ . At each stage of the algorithm, the branch and

**Algorithm** *BranchAndBoundNN*(*TargetQueryLocation: q*, *TargetQueryTime: t*, *RootNode: RN*)

```

begin
  LIST = {RN};
  PessimisticBound = ∞; optimistic(RN) = 0;
  while LIST is non-null do
    begin
      Pick the next node N from LIST based on selection-condition;
      if optimistic(N) ≥ PessimisticBound then prune node N;
      else if node N is a leaf node then
        begin
          L = All objects in N;
          Find position of each object in L at time t;
          find closest object o ∈ L to query location q at t;
          Compute distance D of o to query location q at t;
          If D < PessimisticBound then
            PessimisticBound = D; BestObject = o;
          end
        else begin { Internal Node }
          Find all nodes pointed to by N and denote their pointers by  $N^1 \dots N^r$ ;
          for each node  $N^i$  optimistic(Ni) = ComputeOptimisticBound(Ni, q);
          Add all node pointers  $N^1 \dots N^r$  to LIST;
          end
        Delete node N from LIST;
      end;
    return(BestObject, PessimisticBound);
  end;

```

Figure 1: The Branch and Bound Nearest Neighbor Algorithm

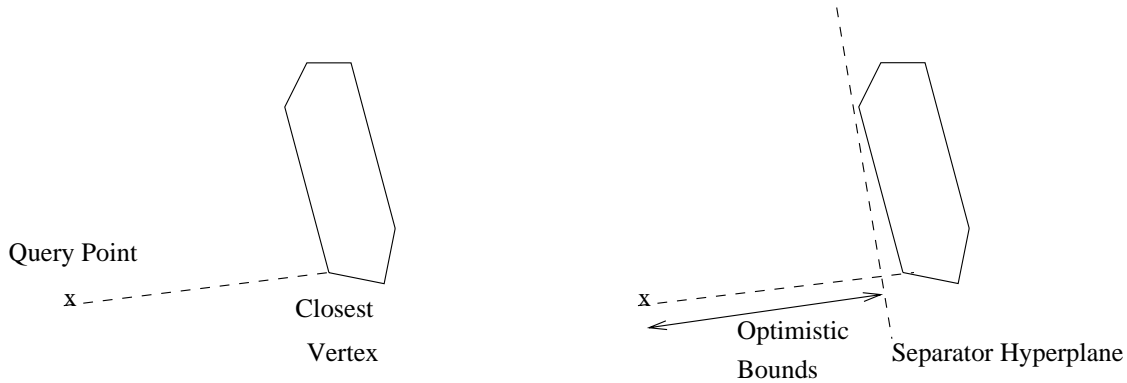


Figure 2: Finding Optimistic Bounds

bound method maintains a global pessimistic bound (denoted by *PessimisticBound*) which is the closest distance to  $q$  of any point encountered so far. The algorithm starts off with initializing *PessimisticBound* to  $\infty$ . The branch and bound method also associates a local optimistic bound  $optimistic(N)$  with each node  $N$ . The optimistic bound  $optimistic(N)$  is a lower bound on the distances at time  $t$  between  $q$  and the objects stored in the subtree rooted at node  $N$ . The details of computing this optimistic bound will be discussed in the next subsection.

In each iteration, a node  $N$  is picked from LIST in accordance with some selection condition (denoted by *selection-condition*). A variety of different selection criteria can lead to either depth-first or breadth-first traversal of the indexing structure. In this paper, we use the selection condition that the node with the smallest optimistic bound is picked for further exploration. After picking the node  $N$ , it is determined if the optimistic bound  $optimistic(N)$  of the node is larger than the global pessimistic bound *PessimisticBound*. If this is indeed the case, then it means that some data point which has already been discovered is closer to  $q$  at time  $t$  than any data point in the subtree rooted at the node  $N$ . Therefore we do not need to further consider the subtree rooted at the node  $N$  and the node  $N$  can be pruned from the set LIST.

If the node  $N$  is not pruned, then it is checked whether it is an internal node or a leaf node. If the node  $N$  is a leaf node then we compute the distance of all objects in the node  $N$  from the query point  $q$  at time  $t$ . We update the *PessimisticBound* and the best object found so far if necessary. On the other hand, if the node  $N$  is an internal node, then we find pointers to all of its children and add them to the set LIST while deleting the node  $N$  itself from the LIST.

The algorithm terminates when the set LIST is null. At this point, all nodes have been either explored or pruned from further consideration. The final value of the variable *PessimisticBound* reflects the true nearest neighbor value.

### 3.1 Finding Optimistic Bounds Efficiently

In this section, we discuss the details of finding a lower bound on the distance between the query point  $q$  and the positions of the objects in the subtree rooted at node  $N$  at time  $t$ . In Figure 1, this is achieved by the function *ComputeOptimisticBound*( $\cdot$ ). The aim is to find an optimistic bound to any parametric representation inside node  $N$  at time  $t$ . Let  $\Gamma^1 \dots \Gamma^m$  vertices of the node  $N$ . Then the snapshot of the vertices of node  $N$  at time  $t$  are given by  $F(\Gamma^i, t)$  for  $i \in \{1, \dots, m\}$ . As discussed above, the procedure *ComputeOptimisticBound* returns a lower bound on the distances between  $q$  and convex hull of  $F(\Gamma^i, t)$  for  $i \in \{1, \dots, m\}$ . We will denote the convex hull of the snapshot representation of node  $N$  by  $N(t)$ . The technique for finding the closest distance to the convex hull of a set of points is well known [14]. However, even this technique can turn out to be quite compute-intensive, especially for trajectories in multiple dimensions. Consequently, we use the following heuristic methodology which returns a looser lower bound while requiring much less time.

Let  $C_N(t)$  be the convex hull of  $F(\Gamma^i, t)$ ,  $i \in \{1, \dots, m\}$ . Our heuristic method relies on the following fact: if we can locate a *separator hyperplane* so that the point  $q$  and  $C_N(t)$  lie on the different sides of this hyperplane, then the distance

between  $q$  and the separator hyperplane is a lower bound on the distance between  $N(t)$  and  $q$ . Given a direction in the space, among all separator hyperplanes which are perpendicular to this direction, the tightest lower bound will be provided by the separator hyperplane which touches at least one corner of the convex hull  $C_N(t)$ . The key in obtaining a good lower bound is to choose the direction judiciously.

To choose the direction, we first perform a linear scan of all vertices of  $N(t)$  to find the vertex closest to  $q$ . Denote the closest vertex by  $F(\Gamma^c, t)$ . Our heuristic method relies on the intuition that the point of  $N(t)$  closest to  $q$  will also be very likely to be close to  $F(\Gamma^c, t)$ . Therefore a good direction for separator hyperplanes would be the line  $l(q, F(\Gamma^c, t))$  joining  $q$  and  $F(\Gamma^c, t)$ .

To find the tightest lower bound along  $l(q, F(\Gamma^c, t))$  we perform a second scan of the vertices of  $N(t)$ . For each vertex, we consider the hyperplane which passes through this vertex and is perpendicular to the line  $l(q, F(\Gamma^c, t))$ . If the point  $q$  and  $N(t)$  lie on the different side of this hyperplane, then the distance between it and the point  $q$  is a candidate for the lower bound. After the linear scan of the vertices, we choose the largest candidate as the tightest lower bound returned by the procedure *ComputeOptimisticBound*( $\cdot$ ).

At first sight, the procedure for computing the optimistic bound might seem a bit expensive. However, this is a main memory operation, whereas the access of the data is achieved from disk. Since disk accesses are orders of magnitude slower than main memory operations, the relative overhead of this computation is small, especially in lower dimensionalities. In order to reduce the overhead further, the best design for the index structure is one in which a flat design of the index is used. In a flat design, each node may contain a multiple pages, and therefore a larger number of records. The overall design of the branch and bound process is simplified further, when a flat design is used. In this case, the nodes are sorted in increasing of the optimistic bounds. The nodes are examined in this order. The procedure is terminated whenever the optimistic bound for a node exceeds the global pessimistic bound.

### 3.2 Correctness

We note that in order to prove that the branch and bound algorithm finds the nearest neighbors accurately, we need to prove that the optimistic bound returned by the algorithm is indeed a lower bound. This ensures that no nodes are pruned falsely, and therefore the final objects found by the algorithm are indeed the nearest neighbors. The key issue is to show that the optimistic distance as computed above to the convex hull  $N(t)$  is a lower bound on the distance to any data point contained inside  $N$  at time  $t$ .

**THEOREM 3.1.** *Let  $S$  be the set of data points in the subtree rooted at the node  $N$ . Then, consider an index structure built on a parametric representation which satisfies the convex hull property. Then, the optimistic distance of the query point  $q$  to the snapshot representation of the convex hull  $N(t)$  of  $N$  is lower bound to the distance of the query point  $q$  to any data point in  $S$ .*

**PROOF.** Let the parametric representations of the corners of  $N$  be denoted by  $(\theta_1^1 \dots \theta_k^1) \dots (\theta_1^m \dots \theta_k^m)$ . Let  $(\theta'_1 \dots \theta'_k)$  be any data point in the subtree rooted at the node  $N$ . We will show that the distance of  $F(\theta'_1 \dots \theta'_k, t)$  to  $q$  is lower than the minimum distance of  $q$  to the convex hull of  $N(t)$ .

We know that since  $(\theta'_1 \dots \theta'_k)$  lies in the subtree rooted at node  $N$ , it must lie inside the convex hull of  $(\theta_1^1 \dots \theta_k^1) \dots (\theta_1^m \dots \theta_k^m)$ . Since the parametric representation satisfies the convex hull property, this means that  $F(\theta'_1 \dots \theta'_k, t)$  lies inside the convex hull of  $F(\theta_1^1 \dots \theta_k^1, t) \dots F(\theta_1^m \dots \theta_k^m, t)$ . Since the euclidean function  $\|\cdot\|$  is quasi-convex, if the query point  $q$  lies outside this convex hull, then the optimistic bound of  $q$  to the convex hull is lower than any point inside it. Therefore, it follows that no node is falsely pruned, and the final object returned is the true nearest neighbor.  $\square$

### 3.3 Extension to other kinds of queries

The technique discussed in this paper can be easily extended to resolving queries such as:

- At a given instant in time  $t$ , find all objects which are within a distance  $d$  of the query point  $q$ .

The query processing technique for this case is similar to that of the nearest neighbor query with a different pruning criterion. In this case, we prune a node  $N$  when the optimistic distance of its snapshot representation  $N(t)$  to  $q$  is larger than  $d$ .

- At a given instant in time  $t$ , find all objects closest to a hyperplane  $l$ .

We refer to this query as a *linear optimization query*. In this case, instead of determining the optimistic distance to a point, we calculate the optimistic bound to the hyperplane  $l$ . However, the process of finding the optimistic bound is slightly different. In this case, it is guaranteed that the closest point is one of the corners of the convex hull. Therefore, only a linear scan through the vertices suffices in order to find the optimistic bound to the hyperplane. We note that such a query can be extended to objects with contours of convex shapes, since the pruning procedure would continue to maintain correctness in that case.

- At a given instant in time  $t$ , find all the objects in the user specified range  $R$ .

The methodology to solve the range query problem uses a similar kind of hierarchical traversal as used for the nearest neighbor query. The difference is that in this case a node  $N$  is considered relevant only if its convex hull  $N(t)$  at time  $t$  intersects with the user specified range  $R$ . We note that if a given node  $N$  does not intersect with the user-specified ranges, then the convex hull property ensures that none of the (enclosed) descendent nodes also intersect this range. Therefore, the descendents of a node are explored only if the bounding rectangle of the snapshot representation of the node intersects with the user-specified range  $R$ . The proof of correctness of this method is similar to that of the case of nearest neighbor queries.

## 4. EMPIRICAL RESULTS

Since there are no known index structures for non-linear objects in arbitrary dimensionality, we do not have a reference point to compare our results with. At the same time, it would be useful to provide some results on the pruning efficiency of the indexing structure resulting from the scheme. In this section, we will discuss some experimental results which illustrate the quality of our indexing scheme. We tested the standard nearest neighbor query, wherein the target point was an object from the same distribution as the remaining data points. We will test the scheme with different kinds of trajectories so as to illustrate its robustness in pruning efficiency.

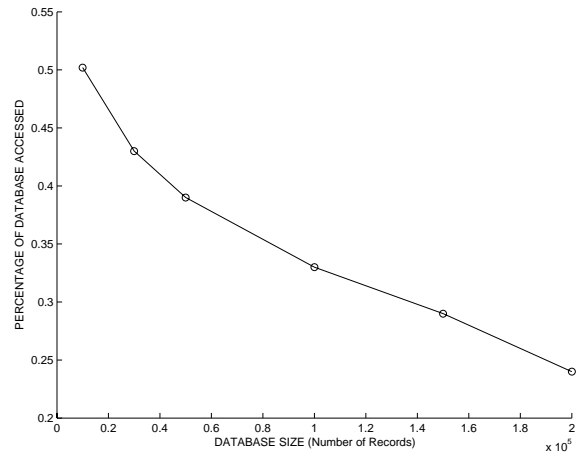


Figure 3: Pruning Performance (Variation with Database Size for 3-dimensional linear trajectory)

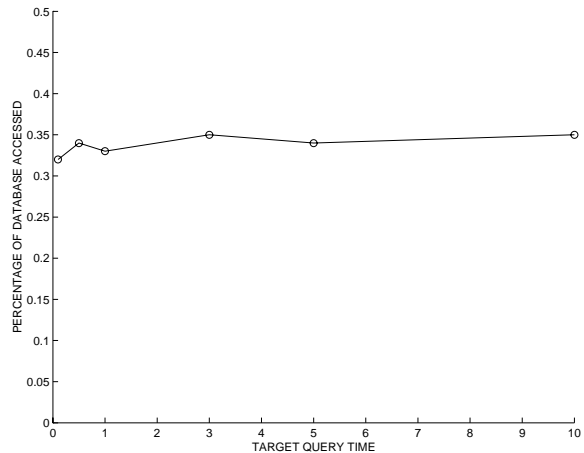
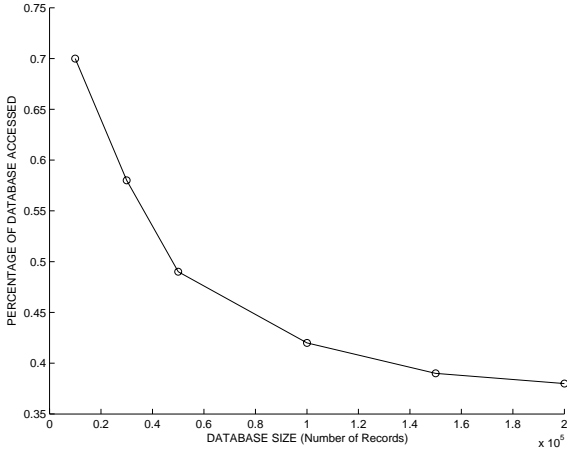


Figure 4: Pruning Performance (Variation with target query time for 3-dimensional linear trajectory)



**Figure 5: Pruning Performance (Variation with Database Size for 2-dimensional parabolic trajectory)**

- Linear Trajectory:** We will first test the scheme with linear trajectories. In each case, we will assume that the parameters of the position and velocity along the  $k$ th dimension are given by  $(\overline{y}_k, \overline{v}_k)$ . Therefore, the position along the  $k$ th dimension at time  $t$  is given by  $\overline{y}_k + \overline{v}_k \cdot t$ . The corresponding parametric representation indexed was  $(\overline{y}_k, \overline{v}_k)$ .
- Nonlinear Trajectory:** We will test a parabolic trajectory in 2-dimensions. We assume that the initial position, velocity, and acceleration along the  $k$ th dimension are given by  $(\overline{y}_k, \overline{v}_k, \overline{a}_k)$ . Therefore, the position along the  $k$ th dimensional at the time  $t$  is given by  $\overline{y}_k + \overline{v}_k \cdot t + \overline{a}_k \cdot t^2/2$ . The corresponding parametric representation was  $(\overline{y}_k, \overline{v}_k, \overline{a}_k)$ .

In addition, we will also test the results from an elliptical trajectory in 2-dimensions. In this case, the functional representation of the trajectory was given by the following:

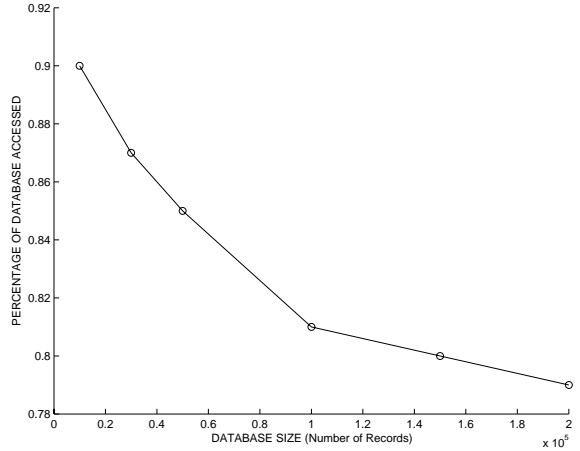
$$x = c_i \cdot \cosine(t) + d_i \cdot \sine(t) \quad (17)$$

$$y = e_i \cdot \sine(t) + f_i \cdot \cosine(t) \quad (18)$$

The corresponding parametric representation indexed was  $(c_i, d_i, e_i, f_i)$ .

The actual generation of the parameters of the trajectory was done using a uniform distribution, such that each of the parameters was independently distributed in the range  $(-b, b)$ . The value of  $b$  used was 1. We note that uniformly distributed data is the worst-case behavior for an indexing structure. This is because the data points are not very well clustered in the uniformly distributed case. In such a case, the pruning is not very efficient. Thus, the results of this paper provide a good understanding of the worst-case quality of indexing of such a method.

In Figure 3, we have illustrated the results on the 3-dimensional linear trajectory by varying the number of points from  $N = 10,000$  to  $N = 200,000$ . The query target was chosen to be a mobile object from the same distribution as the original set of objects and the target time  $t$  was chosen to be equal to 1 in order to determine the query point  $q$ .



**Figure 6: Pruning Performance (Variation with Database Size for 2-dimensional elliptical trajectory)**

Thus, the position of the target object at time  $t = 1$  was used. We have illustrated the average pruning performance over a set of ten queries. The same ten queries were used in each case for consistency. The  $X$ -axis represents the number of points in the database, whereas the  $Y$ -axis represents the percentage of data accessed. It is clear that in each case, only a small percentage of the data is accessed by the algorithm. The other interesting observation is that in each case, the percentage of database accessed reduced with increasing database size. This is a very desirable property, since the indexing problem is most relevant for large databases. The reason for the improvement of the pruning percentage of the branch and bound technique with increasing database size is that for larger databases, the pessimistic bounds are much tighter when the same percentage of the data has been accessed. Thus, nodes are more likely to be pruned at a given stage of the algorithm when larger database sizes are used.

We also tested how the pruning performance varied by varying the query target time. The results are illustrated in Figure 4. The target time was varied from 0.1 to 10. The aim of making this variation was to test how the approach performed for both queries in the near future and queries in the distant future. It is clear from the results of Figure 4, that there was not much variation in the performance of the system for queries of either type. In each case, the pruning performance of the nearest neighbor indexing algorithm continued to be effective.

In Figure 5, we have illustrated the performance of the system on a 2-dimensional parabolic trajectory. The methodology for generating the target query points was the same as in the previous case. We note that at the leftmost end of the graph, only about 0.7% of the database is accessed, whereas at the rightmost end about 0.38% of the database is accessed. Thus, the system shows excellent pruning performance which improves with increasing database size. Similar results were obtained with the use of an elliptical trajectory. The results are illustrated in Figure 6. The amount of data accessed in this case varied between 0.79% and 0.9% of the database size. We note that the results on all of the trajectories show that the nearest neighbor indexing procedure has a performance improves with increasing database size.

This is a very desirable property for developing a practical indexing scheme in a wide variety of large scale applications.

## 5. CONCLUSIONS AND SUMMARY

In this paper we discussed a nearest neighbor indexing scheme for a class of objects with non-linear trajectories in arbitrary dimensionality. We identified a property of non-linear trajectories called the convex hull property and showed that a number of natural trajectories satisfy this property. We showed that this property can be utilized to index a careful representation of the objects so that traditional branch and bound techniques can be applied to the index structure. We also illustrated examples and techniques by which approximate and exact representations satisfying the convex hull property can be found for many important closed-form trajectories. This paper presents the first results for nearest neighbor indexing of non-linear trajectories and thus significantly extends the class of nearest neighbor indexing structures for mobile objects.

## 6. REFERENCES

- [1] P. K. Agarwal, L. Arge, and J. Erickson. Indexing moving points. In *Symposium on Principles of Database Systems* pp. 175–186, 2000.
- [2] N. Beckmann, H.-P. Kriegel, P. Schneider, and B. Seeger. The  $R^*$ -tree: An efficient and robust access method for points and rectangles. In *SIGMOD Conference*, 1990.
- [3] S. Berchtold, D. A. Keim, and H.-P. Kriegel. The X-tree: An index structure for high dimensional data. In *VLDB Conference*, 1996.
- [4] H. D. Chon, D. Agrawal, and A. E. Abbadi. Storage and retrieval of moving objects. In *Mobile Data Management* pp. 173–184, 2001.
- [5] A. Guttman. R-trees: A dynamic index for spatial searching. In *SIGMOD Conference*, 1984.
- [6] N. Katayama and S. Satoh. The SR-tree: An index structure for high dimensional data. In *ACM SIGMOD Conference*, 1997.
- [7] G. Kollios, D. Gunopulos, and V. J. Tsotras. Nearest neighbor queries in a mobile environment. In *Spatio-Temporal Database Management* pp. 119–134, 1999.
- [8] G. Kollios, D. Gunopulos, and V. J. Tsotras. On indexing mobile objects. In *Symposium on Principles of Database Systems* pp. 261–272, 1999.
- [9] E. Kreyzig. In *Advanced Engineering Mathematics*, Wiley, 1988.
- [10] K. I. Lin, H. V. Jagadish, and C. Faloutsos. The tv-tree: An index structure for high dimensional data. In *VLDB Journal*, 1992.
- [11] M. A. Nascimento, J. R. O. Silva, and Y. Theodoridis. Evaluation of access structures for discretely moving points. In *Spatio-Temporal Database Management*, 1999.
- [12] D. Pfoser, Y. Theodoridis, and C. Jensen. Indexing trajectories of moving point objects. In *VLDB Conference*, 2000.
- [13] K. Porkaew, I. Lazaridis, and S. Mehrotra. Querying mobile objects. In *Symposium on Spatial and Temporal Database Management* 2001.
- [14] F. P. Preparata, and M. I. Shamos. In *Computational Geometry: An Introduction*, Springer-Verlag, Germany, 1985.
- [15] S. Saltenis, C. S. Jensen, S. T. Leutenegger, and M. A. Lopez. Indexing the positions of continuously moving objects. In *SIGMOD Conference* pp. 331–342, 2000.
- [16] H. Samet, and G. Hjaltason. Distance browsing in spatial databases. In *ACM TODS Journal*, 1999.
- [17] T. Sellis, N. Roussopoulos, and C. Faloutsos. The R+ tree: A dynamic index for multidimensional objects. In *VLDB Conference*, 1987.
- [18] Z. Song, and N. Roussopoulos. Hashing moving objects. In *Mobile Data Management* pp. 161–172, 2001.
- [19] O. Wolfson, A. P. Sistla, B. Xu, J. Zhou, S. Chamberlain, Y. Yesha, and N. Rish. Tracking moving objects using database technology in DOMINO. In *Next Generation Information Technologies and Systems* pp. 112–119, 1999.
- [20] O. Wolfson, B. Xu, S. Chamberlain, and L. Jiang. Moving objects databases: Issues and solutions. In *Statistical and Scientific Database Management* pp. 111–122, 1998.