PA-Tree: A Parametric Indexing Scheme for Spatio-temporal Trajectories

Jinfeng Ni and Chinya V. Ravishankar

Department of Computer Science and Engineering,
University of California, Riverside
{jni, ravi}@cs.ucr.edu

Abstract. Many new applications involving moving objects require the collection and querying of trajectory data, so efficient indexing methods are needed to support complex spatio-temporal queries on such data. Current work in this domain has used MBRs to approximate trajectories, which fail to capture some basic properties of trajectories, including smoothness and lack of internal area. This mismatch leads to poor pruning when such indices are used. In this work, we revisit the issue of using parametric space indexing for historical trajectory data. We approximate a sequence of movement functions with single continuous polynomial. Since trajectories tend to be smooth, our approximations work well and yield much finer approximation quality than MBRs. We present the PA-tree, a parametric index that uses this new approximation method. Experiments show that PA-tree construction costs are orders of magnitude lower than that of competing methods. Further, for spatio-temporal range queries, MBR-based methods require 20%–60% more I/O than PA-trees with clustered indicies, and 300%–400% more I/O than PA-trees with non-clustered indicies.

1 Introduction

GPS has been widely used for a number of years in support of a variety of new applications, including tracking of vehicle fleets, navigation of watercraft and aircraft, the emergency E911 service for cellular phones [13]. Such applications would benefit greatly from an ability to make complex spatio-temporal queries on databases containing huge amounts of trajectory data about objects moving in two or higher dimensional space.

Work already exists on developing indices to support spatio-temporal queries. Such work is typically either in support of *predictive* queries, which require the future location of objects based on their current locations and velocities (for example, *"find all objects that will be within Union Square in 10 minutes"*), or in support of *historical* queries, which query the past locations of moving objects (for example, *"find all objects which were at the intersection of Freeway 10 and 15 an hour ago"*). In this paper, we focus on historical queries, intended to search a large set of historical trajectories.

In general, we can classify indexing methods into *Native Space Indexing* methods (NSI), and *Parametric Space Indexing* methods (PSI) [18]. In NSI, motion in a d-dimensional space is represented as a series of line segments (or curves) in $d + 1$ dimensional space, using time as an additional dimension. PSI can be regarded as the dual

C. Bauzer Medeiros et al. (Eds.): SSTD 2005, LNCS 3633, pp. 254–272, 2005.

transformation of NSI, where a parametric space defined by the motion parameters is used. PSI has be shown to be an efficient approach for predictive queries, for example, TPR-tree [19], TPR*-tree [22], STRIPES [16].

PSI has not been advocated in the literature for historical queries. Indeed, Porkaew et al. [18] showed that PSI was actually outperformed by NSI for historical queries. Unlike the predicted trajectory case, which uses only one predicted motion function for each object, each historical trajectory could consist of hundreds or even thousands motion functions. PSI will hence introduce large storage overheads, and significantly degrades query performance. As a result, much previous work on historical queries has attempted to index each trajectory in the native space, using approximations such as Minimum Bounding Rectangles (MBRs) [9,8], Octagons [25], or regular grid cells [4]. However, as shown by Kollios et al [10], MBRs are rather coarse approximation for trajectory. A trajectory typically consists of a series of line segments or curves, and does not have any internal area. Consequently, using MBRs may result in a large amount of dead space, leading to a significant loss in pruning power.

1.1 Our Work

In this paper, we revisit the issue of indexing historical trajectory in parametric space. Unlike previous work in the area [18], we do not represent each line segment or curve with a parametric function. Instead, we try to *approximate* a series of line segments or curves with *a single* continuous polynomial. This approximated trajectory may not perfectly match the original trajectory. However, if we also keep track of the maximum deviation between the approximation and the original movement, we can still ensure that the approximation is *conservative*, and will not generate false negatives. Therefore, as long as the maximum deviation is small, the approximated polynomial function and the maximum deviation together provide a much tighter approximation than the generally used MBRs. We are therefore able to improve query performance significantly.

The fundamental observation behind our scheme is that trajectories, in general, have a certain degree of smoothness, as suggested in [3]. First, object movements are governed by the laws of physics, resulting in smooth motion trajectories. Second, many objects are constrained to move along road networks, which usually have some degree of smoothness. Indeed, for *similarity-based queries*, exploiting the smoothness of trajectories has yielded performance far better than that of previous methods[3].

The work in [3] also uses polynomials to approximate trajectories, but there are major differences between our work and theirs. First, [3] targets similarity-based queries, and defines similarity over entire trajectories of equal length, ignoring the time components. Hence the techniques in [3] are generally not applicable to spatio-temporal databases, where the time component is crucial in answering timestamp or time interval queries [17]. Second, the lower-bound lemma in [3] is only valid for similarity queries, so that other approaches are needed to deal with spatio-temporal queries using polynomial approximations. Further, [3] uses approximations of the same degree for all the trajectories, which can cause serious difficulties when the approximation degree is high. In contrast, we use different polynomials of different degrees for different trajectories, and develop a two-level index structures to avoid this problem.

In this work, we make the following contributions:

- We revisit the issue of indexing historical trajectory in parametric space. Observing the smoothness of object movements, we show that parametric indexing using polynomial approximations can improve query performance significantly over current schemes using native space indexing.
- We develop a cost model to optimize the degree of the polynomial approximation given a trajectory segment. Further, we present the PA-tree, a new index scheme for historical trajectory data, based on polynomial approximations.
- We evaluate the performance of our schemes using synthetic trajectory datasets. Our empirical results indicate that in most cases, the MVR-trees require 20% - 60% more IO than PA-trees with clustered indicies, and 300%–400% more IO than PA-trees with non-clustered indicies. More importantly, the cost of constructing PA-trees is orders of magnitude faster than the construction of MVR-trees, suggesting that PA-trees may be suitable for on-line indexing of trajectories.

2 Related Work

MBRs have been widely used to approximate multi-dimensional data, and consequently R-trees are the most common index structure for multidimensional data. Earlier work using MBRs for trajectories includes the RT-tree [24] and 3D R-tree [23]. However since the RT-tree does not take temporal attributes into account during the insertion/deletion, timestamp or time interval queries are inefficient. 3D R-tree is inefficient for timestamp queries, since the query time depends on the total number of entries in the history [21].

Kollios et al. [11] present methods for indexing linear historical trajectories. They model a long-lived trajectory with multiple MBRs by splitting it into segments to reduce the large dead space resulting from the use of a single MBR, and use partial-persistent R-trees ("PPR-tree") to index the multiple MBRs. This work is extended in [9,8], where the motion function could be arbitrary (In the latest work [8], term "MVR-tree" is used in stead of "PPR-tree"). This method can be more efficient than 3D R-tree, since the total empty volume after splitting would be reduced. However, since this method still uses MBRs for approximating each segment, there remains significant dead space.

Zhu et al [25] used octagonal prisms, which are MBRs whose four corners are cut off to approximate trajectory. However, their experiments demonstrate only small differences between octagonal prisms and MBR when the number of splits increases to a certain point, since little gains will result from cutting off MBR corners when the number of splits becomes large.

Some previous work has been based on a discrete event model, under which an object is assumed to stay at its current position until it issues an update to the server. However, this model can not be used to represent gradual changes in object locations, limiting its applicability[4]. The basic idea is to build a separate R-tree for each timestamp, as in HR-tree [14] and MR-tree [24]. Unchanged nodes are not duplicated in consecutive R-trees to reduce the storage cost. However, these index structures are only efficient for timestamp queries, but are not efficient for time interval queries [4,21]. The MV3R-tree [21] is a hybrid structure that uses a multi-version R-tree for timestamp

queries, and a small 3D R-tree for time-interval queries. The two indices share the same leaf pages, in order to reduce the storage cost, resulting in a quite complex algorithm for maintaining the indices [4].

SETI [4] is an indexing method which can support both inserts and searches. SETI uses two-level index structures to decouple the spatial and the temporal dimensions. Space is partitioned into multiple cells, and the temporal attributes of all line segments intersecting a cell will be indexed with a 1-dimensional index structure. However, since multiple line segments of the same trajectory may overlap the query range, SETI must eliminate duplicates, which may be expensive. Also SETI does not have the *trajectory preservation* property [17], since each data page may contain segments of multiple trajectories, with no guarantee all line segments of one trajectory will be stored together. Hence, SETI may not be able to efficiently support trajectory-based queries [17].

Polynomial approximations have been used to approximate *predictive* trajectories by Tao et al. [20], who use *STP-trees* to index the polynomial coefficients. Several major differences exist between their work and ours. First, our query types are different. Second, [20] applies the same degree of approximation for all trajectories, assuming the same motion type for *all* objects. In practice, different objects may have trajectories with different complexities. In contrast, we choose the degree of polynomial approximation based on the complexities of trajectory, a strategy applicable in more general scenarios. Further, in [20], when a k degree polynomial is used for each axis in a d-dimensional space, the STP-tree becomes an index structure in the parametric space of $(k+1)d$ dimensions, leading to the problem of curse of dimensionality for large k. Unlike [20], we adopt a two-level structure (see Section 6) to address this problem.

3 Problem Definition and Data Model

Data Model. In many location-based services, location data are obtained by periodic sampling. Specifically, the trajectory for an object O_i has the form

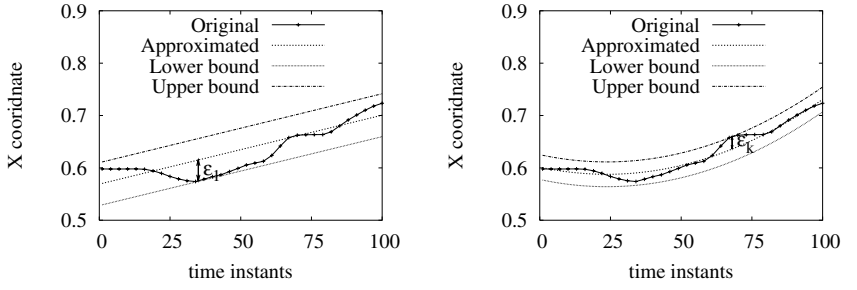
$$\mathit{Trj}(O_i) = \{\mathit{ID}_i, t_0, t_1, \cdots, t_n, f_1(t), f_2(t), \cdots, f_n(t)\}$$

Function $f_j(t)$ is a movement function representing movement during time interval $[t_{j-1} : t_j]$, $1 \leq j \leq n$. The interval $[t_0 : t_n]$ is the *lifetime* of the trajectory.

Our approach is applicable to any movement function $f(t)$, as long as we can determine the location of the object at any time instant during its lifetime from $f(t)$. For simplicity of exposition, we adopt a linear mobility model, which is widely used in the literature [4,17]. Each $f_j(t)$ is now a linear function of time, so that a trajectory consists of a series of connected line segments. This representation is refereed to as a *polyline*.

As in previous work [9,8], we assume time is discrete, and the dataset temporal range $[0, T]$ contains the lifetimes of all the trajectories. We assume an object moves in a two-dimensional XY-space. The extension to higher dimensions is straightforward.

Query Types. We focus mainly on historical coordinate-based queries, in particular on spatio-temporal range queries, since spatio-temporal range queries are essential building blocks for all other types of queries. A spatio-temporal range query may be a timestamp query, or a time interval query. A timestamp query $Q(r, t)$ asks for all the objects



(a) Linear approximation. (b) Polynomial approximation.

Fig. 1. Approximating a trajectory segment with polynomials

within spatial range r at timestamp t. Similarly, a time interval query $Q(r, t_b, t_e)$ asks for all the objects which were within spatial range r at any timestamp $t \in [t_b : t_e]$.

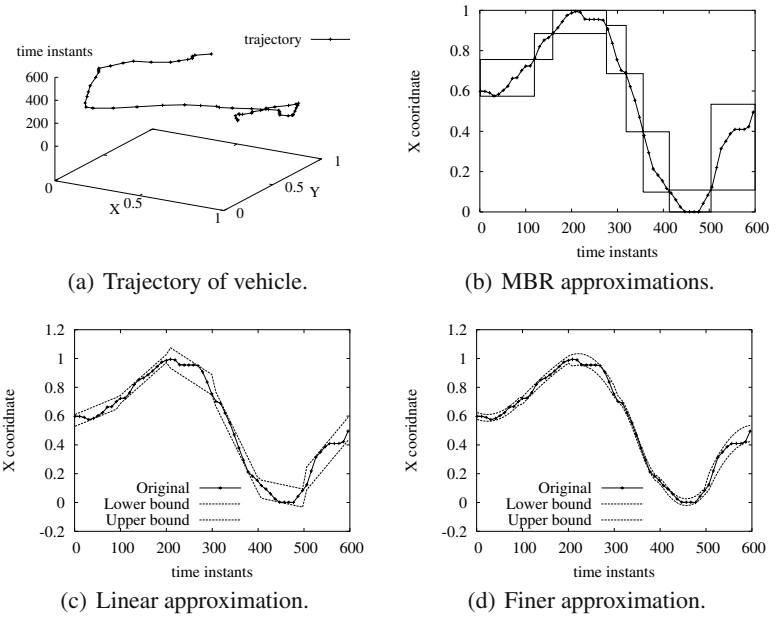
The PA-tree also supports efficient execution of *trajectory-based queries*, which may take the output of coordinate-based queries as input and retrieve the exact trajectory so that certain properties, such as direction or speed can be derived [17,25]. As we will explain, the PA-tree allows a series of consecutive line segments belonging to the same trajectory to be stored together. This *trajectory preservation* property ensures the trajectory-based queries can be answered efficiently with the PA-tree.

4 Overview of Our Approach

Our approach proceeds in two steps. In the first step, we calculate the parametric representations used to approximate each trajectory. We will approximate a trajectory in the XY-space with two polynomial functions: $\hat{f}^x(t)$ and $\hat{f}^y(t)$ modeling movement in the X direction and in the Y direction, respectively, where t is time. We also determine the maximum deviation of the polynomial approximation from the exact movement in X and Y dimensions. The polynomial coefficients and the maximum deviation suffice for us to make the approximation conservative, guaranteeing no false negatives.

Fig. 1(a) and Fig. 1(b) shows the X-component of a trajectory, and illustrates how we construct a linear and an order-k polynomial approximation to it. Such approximations are not exact, so we create conservative upper and lower bounds for the object's position by offsetting the approximating polynomial upwards and downwards by an amount equal to the maximum deviation between the trajectory and the polynomial. We can now guarantee that the object will be located within these bounds.

In the second step, we build an index structure over the coefficients obtained in the first step. However, not all trajectory are likely to be equally complex, so that we may need polynomials of different degree for different trajectories. This causes problems when building an index structure using the coefficients, since the dimensionalities of the indexed items may be different. Current index structures assume that the dimensionality of all data is the same. Adopting the same polynomial degree for all trajectories is not advisable, since the curse of dimensionality will quickly degrade the performance of any index structure in high dimensional space.



(a) Trajectory of vehicle.

(b) MBR approximations.

(c) Linear approximation.

(d) Finer approximation.

Fig. 2. Comparison of approximations for a moving vehicle trajectory collected in [1]

Two-Level Indexing. We address this problem by using a two-level index structure. The first-level index structure uses only the first two coefficients of each polynomial, so that each data entry is a 6-tuple (two coefficients for each dimension, and the corresponding maximum deviations). This strategy ensures that we are not operating in a high dimensional space, so that an R-tree or its variants can still be efficient for indexing. As we will illustrate in Section 5, by appropriately splitting the temporal domain $[0, T]$ into intervals, we can adopt a piecewise linear approximation in the first level index structure, each linear approximation corresponding to multiple line segments in the trajectory. However, even with this piecewise-linear approximation, we can achieve much smaller dead space than the MBRs with the same size of representation.

The second-level index structure is elaborated within the leaf nodes of the first-level structure. As noted earlier, some trajectories may be complex and require higher-degree polynomial approximations. The higher-degrees coefficients are stored in the second level structure. If we descend to the leaf nodes in the first level structure, and still are unable to determine whether the trajectory satisfies the query predicates, the additional coefficients can be retrieved and used in the filtering step. As our experiments will show, most trajectories can be approximated very well with quadratic or cubic polynomials, so that the second level structure does not introduce significant space overhead.

An Illustrative Example. Figure 2(a) plots the trajectory of a moving vehicle for 10 minutes, collected in the Intellishare project [1] at the University of California–Riverside. Figure 2(b) plots the X-movement against time, and the eight MBRs obtained with the LAGreedy algorithm proposed in [9,8]. We note that the eight MBRs together requires $8 \times 6 = 48$ values. Figure 2(c) plots the result of our method, in

which the trajectory is split into 6 segments, each of which is approximated with a linear function. Each segment requires two coefficients, and one maximum deviation each for X-movement or Y-movement, plus the temporal intervals. In all, 48 values are required for the approximations. It is quite clear that our polynomial approximations produce much smaller dead space than the MBR approximations. We should note that since we split all the trajectories at the same split timestamps, there is no need to keep the temporal intervals in the intermediate nodes in the index structure, which would further save storage cost. Figure 2(d) plots the approximation with more coefficients, with significantly reduced dead space. This example motivates our work.

5 Approximating Trajectories with Polynomials

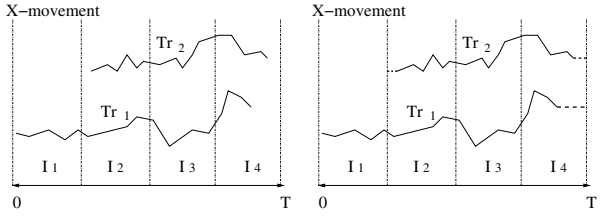
In this paper, we propose an approximation in parametric space by using Chebyshev polynomials. Chebyshev polynomials have been shown to have the near-optimal L_∞ deviation among all approximations with the same degree [12], and perfectly match our requirements. Further, the Chebyshev coefficients are easy to compute [12,3].

We have chosen to split each trajectory into multiple segments by dividing the temporal domain $[0, T]$ into m disjoint time intervals, each of which is approximated with a polynomial. There are two reasons for such splitting. First, approximating the entire trajectory with a single polynomial may require a polynomial of high degree, leading to a high-dimensional indexing problem. Second, the marginal benefit for the first few coefficients will be much larger than that of high-order coefficients. Therefore, it is wise to split the trajectory into multiple segments, and approximate each segment with a lower-degree polynomial.

5.1 Splitting the Time Domain

We split the temporal domain $[0, T]$ into m equal time intervals: $I_1 = [0, T_1), I_2 = [T_1, T_2), \cdots, I_m = [T_{m-1}, T)$, where $T_1, T_2, \cdots, T_{m-1}$ are called splitting timestamps. Each trajectory is split into multiple segments using the same $m - 1$ splitting timestamps. This strategy is different from that in [9], where each trajectory selects different splitting timestamps. We choose our strategy for three reasons. First, since we index in parametric space, a set of segments can not be clustered unless they have the same temporal domain, since it would be meaningless to cluster coefficients corresponding to different temporal domains. Second, even with a equal-sized splitting strategy, we can still use different numbers of coefficients for different trajectories. Indeed, basing the number of coefficients for approximation on the trajectory complexity is equivalent to using different splitting timestamps. Finally, using equal-length splitting intervals obviates the need to maintain time intervals in index nodes. This could significantly reduce the storage cost of the index structure, and eventually lead to a reduction of I/O cost during the filtering step.

One problem in using equal-sized intervals is that some trajectories may begin or end within some time interval. For instance, in Figure 3(a), trajectory Tr_2 begins in the middle of interval I_2. We can simply extend its lifetime to the beginning of I_2, and require that the object remain at its initial location during this extension. This will



(a) Split temporal domain into intervals. (b) Extend the lifetime of trajectories.

Fig. 3. Splitting the temporal domain

result in a "faked" trajectory. However, as long as we maintain the actual object lifetime in the second level index structure, we will see no false positives. Figure 3(b) shows the trajectories with extended lifetimes, where the dashed line segments represents the extension.

An important issue is how to choose the number of intervals. This is a complex problem, since we have to minimize the overall query cost. Similar as [8,4] which chose the number of MBRs or grid cells through experiments, we also vary the number of intervals in experimental evaluations, and choose the number of intervals which result in the best query I/O cost.

5.2 Approximating a Trajectory Segment with a Polynomial

We now consider how to obtain polynomial approximation (PA) with Chebyshev polynomials. Consider a trajectory segment in the temporal interval $I_i = [T_{i-1}, T_i)$

$$\{ID_j, t_0, t_1, t_2, \cdots, t_s, f_0^x(t), f_0^y(t), f_1^x(t), f_1^y(t), \cdots, f_{s-1}^x(t), f_{s-1}^y(t)\}$$

where the linear functions f_i^x and f_i^y describe the X-movement and the Y-movement during the interval $[t_i, t_{i+1}]$, and $t_0 = T_{i-1}$, $t_s = T_i$. We illustrate the approximation for the X-movement only, so we will omit the superscript x when no confusion can arise. We can rewrite piecewise linear functions for X-movement in functional form as follows: $f(t) = f_i(t),$ if $t \in [t_i, t_{i+1}]$.

Given function $f(t)$, we can use Chebyshev polynomials as the base functions to get the approximated function $\hat{f}(t)$. We first normalize the temporal domain $[t_0, t_s]$ to the interval [-1,1], by substituting $t' = \frac{2t - t_s - t_0}{t_s - t_0}$. Now, $f(t)$ can be approximated as

$$\hat{f}(t) = c_0 T_0(t) + c_1 T_1(t) + \cdots + c_k T_k(t), \tag{1}$$

where $T_i(t) = \cos(i \arccos(t)), t \in [-1, 1]$ is the Chebyshev polynomial of degree i, and the coefficients $c_0, c_1, \cdots, c_k$ are to be determined. The Gauss-Chebyshev formula leads to the following theorem [12], which gives an explicit way to compute the coefficients:

Theorem 1. *Let $f(t)$ be the function over interval $[-1, 1]$ to be approximated. The polynomial $T_m(t)$ has m roots $\rho_j = \cos\frac{(j-0.5)\pi}{m}$ for $1 \leq j \leq m$. Now,*

$$c_0 = \frac{1}{m}\sum_{j=1}^{m} f(\rho_j)T_0(\rho_j) = \frac{1}{m}\sum_{j=1}^{m} f(\rho_j)$$
$$c_i \quad = \frac{2}{m}\sum_{j=1}^{m} f(\rho_j)T_i(\rho_j),\ 1 \leq i \leq k$$

Since $\hat{f}(t)$ only approximates $f(t)$, it may differ from $f(t)$ at time instant $t \in [t_0, t_s]$. To ensure that this approximation leads to no false negatives, we must find a conservative approximation such that the approximation is guaranteed to contain the object's location at all times. This goal can be achieved by computing the maximum deviation $\epsilon_k = \max\left\{|f(t) - \hat{f}(t)|\right\}, t \in [t_0, t_s]$, after obtaining the $k+1$ coefficients. Now the range $[\hat{f}(t) - \epsilon_k, \hat{f}(t) + \epsilon_k]$ is guaranteed to contain $f(t)$ for $t \in [t_0, t_s]$.

The $k + 1$ coefficients can be computed in time $O(mk)$, where m is the highest degree of Chebyshev polynomial used in the approximation, and k is the number of coefficients. The computation of maximum deviation error requires time $O(lk)$, where l is the number of instants between t_0, t_s. Therefore, the total cost is $O(mk + lk)$.

5.3 Clustering Multiple Polynomials

As with any index, each level of the PA-tree must maintain a bound on the key attributes of lower-level nodes. In our case, each non-leaf entry in the index structure for interval $I_i = [T_{i-1}, T_i)$ maintains certain coefficients $\{c_i^\top\}$ and $\{c_i^\perp\}$ that enable us to compute conservative upper and lower bounds for the values of the polynomials $\hat{f}_1(t), \hat{f}_2(t), \cdots, \hat{f}_n(t)$ stored in the child node pointed by the entry. We will now discuss how to compute these bounds if we are using order-k polynomial approximations. In Section 6.2 we discuss the use of these coefficients in query processing. As in Section 5, we first normalize t to $[-1, 1]$.

Let $\hat{f}_j(t) = c_{0,j} + c_{1,j}T_1(t) + \cdots + c_{k,j}T_k(t)$, where $1 \leq j \leq n$. We store the values $c_i^\top = \max\{c_{i,j}\}$, $c_i^\perp = \min\{c_{i,j}\}$, where $1 \leq j \leq n$, $0 \leq i \leq k$ in the index node. For any $t \in [-1, 1]$, the lower- and upper-bounds are computed from the functions

$$\phi^\perp(t) = \ c_0^\perp + a_1(t)T_1(t) + \cdots + a_k(t)T_k(t), \text{ and}$$
$$\phi^\top(t) = c_0^\top + b_1(t)T_1(t) + \cdots + b_k(t)T_k(t), \text{ where}$$

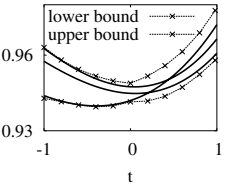


Fig. 4. Bounds

$$a_i(t) = \begin{cases} c_i^\top, & \text{if } T_i(t) \leq 0 \\ c_i^\perp, & \text{otherwise,} \end{cases} \quad \text{and } b_i(t) = \begin{cases} c_i^\perp, & \text{if } T_i(t) \leq 0 \\ c_i^\top, & \text{otherwise,} \end{cases} \quad \text{for all } i.$$

Theorem 2. *The bounds $\phi^\perp(t)$ and $\phi^\top(t)$ are conservative.*

Proof. Consider any $\hat{f}_j(t) = c_{0,j} + c_{1,j}T_1(t) + \cdots + c_{k,j}T_k(t)$. We will have $\phi^\top(t) - \hat{f}_j(t) = \sum_i (b_i(t) - c_{i,j})T_i(t)$, and $(b_i(t) - c_{i,j}) > 0$ when $T_i(t) > 0$, and $(b_i(t) - c_{i,j}) < 0$ when $T_i(t) < 0$, from $b_i(t)$'s definition. Now, $\phi^\top(t) - \hat{f}_j(t)$ is the sum of positive terms, and is positive. The proof for $\phi^\perp(t)$ is similar.

Figure 4 shows three solid curves representing three 3-order polynomials, while the dotted lines represent the lower- and upper-bounding polynomials computed as above. We note that this bound may not be tight for all t, but it is conservative, guaranteeing that any polynomial $\hat{f}_j(t)$ will be inside the bound. There is an issue with a possibly high computation cost when the query interval $[t_b, t_e]$ is large, since we may have to compute the bound for all $t \in [t_b, t_e]$. Fortunately, in the first level of the PA-tree, only the linear approximations in the form of $c_o + c_1 T_1(t) = c_0 + c_1 t$ are used, which has a much simpler way to compute the bound over $[t_b, t_e]$, due to the monotonicity of $c_0 + c_1 t$ (see Section 6.2).

5.4 Comparing Approximation Quality

To gauge the potential for improvement with our scheme, we compare the dead space obtained using our method with that obtained with the MBR approximation. This metric captures the pruning power of index structures based on the respective approximations. Larger amounts of dead space would suggest smaller pruning power, since it will result in more refinement candidates.

We compare our scheme with the MBR approximations obtained using the LAGreedy algorithm [9]. The volume of each MBR, is simply the product of the edge lengths along the X-dimension, Y-dimension and the temporal dimension. Each entry is a 6-tuple, as discussed in Section 4.

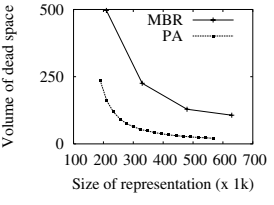


Fig. 5. Dead space

If we use $k+1$ coefficients each to approximate the X-movements and Y-movements, the volume of dead space can be computed as $4\epsilon_k^x \epsilon_k^y (t_s - t_0)$, where $[t_0, t_s]$ is the temporal domain. The representation size is $2(k+1)+5$, since we represent $2(k+1)$ coefficients in all, the value of k, as well as the maximum deviation and the temporal domain.

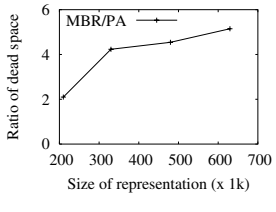


Fig. 6. Ratio

Figure 5 and 6[1] compare the quality of our polynomial approximation with that of MBR approximations for a dataset of 5000 trajectories generated using the network-based generator of [2]. The generator took the road network in San Joaquin County, CA as its input and simulated the movements of objects moving along the road network (see Section 7). Clearly, we see that for a given representation size, the dead space with our our polynomial approximation is much as to 2–5 times smaller than the dead space with MBR approximations. This is expected, since the polynomial approximation captures the inherent smoothness of the movement, and treats the tra-

[1] The ratio is computed as the dead space of MBRs over the dead space of PA with the same or smaller size of representation.

jectory as a polyline, rather than a spatial object with extent. Therefore, there is significant potential for improving the overall query performance.

5.5 Choosing the Degree of Approximating Polynomials

In general, the polynomial degree k should be determined based on the characteristics of trajectories. Clearly, there is a trade-off between the approximation quality and the degree k used for approximation. A smaller k value requires less space in the index, as well as less I/O during the filter step. On the other hand, fewer coefficients may result in poorer filtering, causing more trajectories to be examined during the refinement step, increasing its I/O cost.

Another consideration is the complexity of the trajectory segment. Obviously, if the trajectory segment has a relatively simple form, a few coefficients will suffice to get small deviation error. However, since we are not aware of any well-defined notion of complexity for this context, it is not easy to estimate the optimal degree.

We present a heuristic method to estimate the degree k, aimed at minimizing the expected size of representations to be retrieved during query evaluation. We will make the following reasonable assumptions. First, we assume the spatial range r of the query is an $l_x \times l_y$ rectangle, with l_x and l_y uniformly distributed between 0 and some maximum value L. Second, the spatial range r itself is uniformly distributed in the region normalized to a unit square.

Let S_k be the size of representation of k-degree polynomials approximation. Let S be the size of the exact representation for the trajectory segment. Next, we derive the expected size of representations that have to be retrieved for a random query.

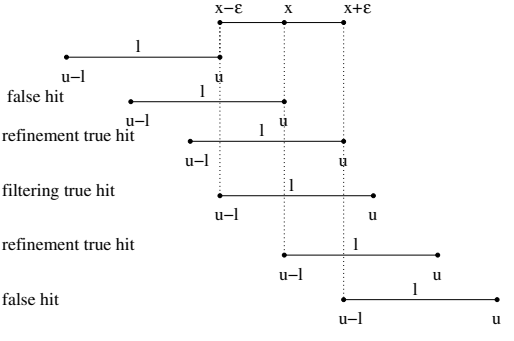


Fig. 7. True and false hits

If the approximation does not intersect the query range, we can safely prune it out during the filter step. If a segment's approximation lies *completely* inside r, we can safely say it is a true hit during the filter step, and no further checking is needed. We call this category of true hit a *filtering true hit (FT)*. Otherwise, the segment becomes a *candidate (CD)* for the refinement step, in which its exact representation must be retrieved. If a segment lies outside r, we have a *false hit (FH)*. If a segment truly lies inside r, we will record a true hit during refinement, and refer to it as a *refinement true hit (RT)*.

To estimate the expected I/O cost, we must estimate the probability that the trajectory segment is a candidate for refinement. A candidate can be either a false hit or a refinement true hit. In the following, we consider the X and Y dimensions separately, and omit the x and y superscripts and the subscript k when no confusion is likely.

Let the query range r's projection on X-dimension be $[u - l, u]$. Let the exact location of an object on a trajectory segment at query timestamp be x. As shown in Figure 7,

if $u \in [x-\epsilon, x) \cup (x+l, x+l+\epsilon]$, we have a false hit, since r will overlap with our trajectory approximation, but x does not belong to $[u-l, u]$. Therefore, the probability a false hit on the X-dimension is $\Pr[FH_X] = 2\epsilon$.

Further, as shown in Figure 7, if $l \leq 2\epsilon$, there can be no filtering true hit, since the segment's approximation along the X-dimension can not be completely inside r's project on X-dimension. However, when $l \geq 2\epsilon$, and $u \in [x+\epsilon, x+l-\epsilon]$, we have a filtering true hit on the X-dimension. When $u \in [x, x+\epsilon] \cup [x+l-\epsilon, x+l]$, we have a refinement true hit. Therefore, the probability of a refinement true hit on X-dimension is $\Pr[RT_X] = \frac{1}{L}\left(\int_0^{2\epsilon} l dl + \int_{2\epsilon}^{L} 2\epsilon dl\right) = 2\epsilon - 2\epsilon^2/L$. Now, the probability the trajectory segment is a candidate for refinement on X-dimension is $\Pr[CD_X] = \Pr[FH_X] + \Pr[RT_X] = 4\epsilon - 2\epsilon^2/L$.

Now, since a candidate occurs only when it is a candidate on the X- or Y-dimensions, the probability of a candidate is $\Pr[CD] = 1-(1-\Pr[CD_X])(1-\Pr[CD_Y]) = 1-(1-4\epsilon_k^x + 2(\epsilon_k^x)^2/L)(1-4\epsilon_k^y + 2(\epsilon_k^y)^2/L)$.

Now, the expected I/O cost for the trajectory segment when using degree k for approximation is $\overline{IO}_k = S_k + \Pr[CD]S$. This metric provides us a heuristics for estimating the degree k required to be used in the polynomial approximation. More specifically, we would like to find k such that $\overline{IO}_k$ is minimized.

6 PA-Trees and Query Processing

We now present the PA-tree, a new method for indexing polynomial approximations of 2-D trajectories. PA-trees resemble R*-trees, but each entry consists of polynomial coefficients, rather than MBRs. We recall that the temporal domain $[0, T]$ is split into m intervals. In a gross sense, the root node of a PA-tree has m index trees as children, each responsible for indexing trajectory segments within one of these intervals.

Figure 8 shows a PA-tree. Indexing in PA-trees actually occurs at two levels. The first level of indexing is an R*-tree like structure, and is used to index the two leading coefficients of the polynomial describing movement along each dimension. It is reasonable to see this as a 4-dimensional indexing problem, with each dimension corresponding to one coefficient. Each entry in the index structure also holds the maximum deviation errors ϵ_1^x and ϵ_1^y.

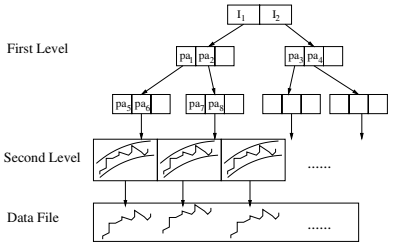


Fig. 8. An example of PA-tree

As in R*-trees, an entry in a leaf node has the form (ptr, pa), where ptr is the pointer to the exact representation of the trajectory segment, and pa is a tuple of 6 values: $\langle c_0^x, c_1^x, c_0^y, c_1^y, \epsilon_1^x, \epsilon_1^y \rangle$. Entries in non-leaf nodes are of the form (ptr, pa), where ptr is the pointer to a child node, and pa has the form $\langle c_0^{x\perp}, c_0^{y\perp}, c_1^{x\perp}, c_1^{y\perp}, c_0^{x\top}, c_0^{y\top}, c_1^{x\top}, c_1^{y\top}, \epsilon_1^x, \epsilon_1^y \rangle$, representing the lower (upper) bounds of the coefficients for the entries stored in the child node pointed by ptr. Also, pa maintains the maximum ϵ_1^x and ϵ_1^y for all the entries in the subtree.

In the second level, we store more coefficients as well as the corresponding maximum deviation for each trajectory segment, if the estimated degree is larger than 1 (See subsection 5). This information provides more pruning power than the linear approximation used in the first level structure.

Insertions and deletions are similar to the corresponding operations for R*-tree. The primary difference is that we need to ensure that the $\epsilon_1^x, \epsilon_1^y$ values in the non-leaf nodes are the maximum $\epsilon_1^x, \epsilon_1^y$ for all the segments in its subtree.

6.1 Improving Query Performance with Clustered Indices

As suggested in [9], clustered indices can significantly reduce the I/O cost for the refinement step. This optimization can also be applied to the PA-tree, so that all data associated with a leaf node entry is stored sequentially on the disk next to the leaf node itself, resulting in sequential retrieval of data. Clustered indices can be created in two steps. In the first step, a non-clustered index is created. In the second step, we can reorganize the disk pages to store data pages sequentially next to the leaf pages.

We note that clustered indices may not be an appropriate choice in some applications. For example, some applications may need indices clustered on other attributes, say object ID. Also, some applications that may already have collected large amounts of trajectory data, may not allow data reorganization due to its high cost. Consequently, we consider both clustered and non-clustered indices in our experimental evaluation. For both cases, PA-tree shows significant improvements over current methods.

6.2 Query Processing

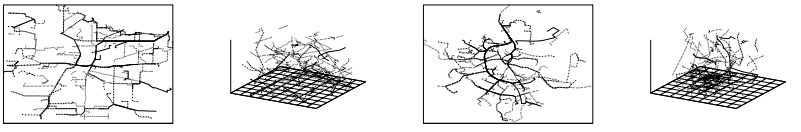
Given a query $Q(r, t_b, t_e)$, we start with PA-tree root which contains the pointers to the segment index roots and the corresponding temporal intervals. We check whether the temporal interval intersects $[t_b, t_e]$. If they do not, the subtree rooted at that root node is discarded. Otherwise, we search the corresponding subtree.

Let $I_i = [T_{i-1}, T_i]$ be the temporal interval corresponding to an entry in a non-leaf node in the PA-tree. Given $Q(r, t_b, t_e)$, we must check whether there is a trajectory segment inside r at any time $t \in [\max\{t_b, T_{i-1}\}, \min\{t_e, T_i\}]$. Let the index entry be $\langle c_0^{x\perp}, c_0^{y\perp}, c_1^{x\perp}, c_1^{y\perp}, c_0^{x\top}, c_0^{y\top}, c_1^{x\top}, \epsilon_1^x, \epsilon_1^y \rangle$. In the following discussion, we will omit the superscripts x and y for the sake of clarity.

As in Section 5, t is first normalized to $[-1, 1]$. Let t_1 and t_2 be the normalized values of $\max\{t_b, T_{i-1}\}$ and $\min\{t_e, T_i\}$, respectively. Now, the non-leaf entry represents all movement in the approximated linear form $c_0 + c_1 T_1(t) = c_0 + c_1 t$, where $c_0 \in [c_0^\perp, c_0^\top]$, and $c_1 \in [c_1^\perp, c_1^\top]$. In principle, we can apply the dual transformation technique of [10] to check whether there are linear trajectories intersecting r during $[t_1, t_2]$. However, the slope c_1 and the temporal attribute t could be either positive or negative, making it hard to apply duality transformations. Instead, we determine the upper and lower bounding polynomials for the motion segment in the form $c_0 + c_1 t$, where $c_0 \in [c_0^\perp, c_0^\top]$, $c_1 \in [c_1^\perp, c_1^\top]$, and $t \in [t_1, t_2]$. If ϵ_1 is the maximum deviation error, we use the monotonicity of $c_0 + c_1 t$ to compute the lower bound as: $x^\perp = c_0^\perp + \min\{c_1^\perp t_1, c_1^\perp t_2, c_1^\top t_1, c_1^\top t_2\} - \epsilon_1$, and the upper bound as:

Table 1. Characteristics of the datasets used in the experiments

Dataset	Description	Total objects	Average movement functions per object	Total num. of line segments	Dataset size (MB)
CA5k	San Joaquin, CA	5,000	300	1,500,000	48
OD5k	Oldenburg	5,000	258	1,290,000	41



(a) CA5k. (b) CA5k(3d). (c) OD5k. (d) OD5k(3d).

Fig. 9. A snapshot of datasets

$x^{\top} = c_0^{\top} + \max\{c_1^{\perp}t_1,\, c_1^{\perp}t_2,\, c_1^{\top}t_1,\, c_1^{\top}t_2\} + \epsilon_1$. If the computed range intersects with the query range r, we know there may be candidates satisfying the query predicates. We now descend the tree and repeat the process for the subtree rooted at this entry, down to the leaf nodes.

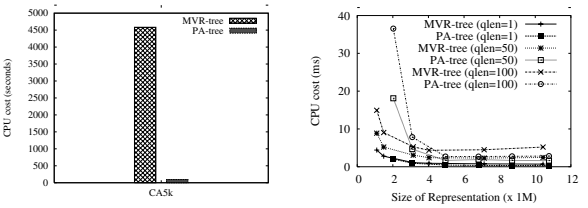
At the leaf node, we will first retrieve the $k+1$ coefficients in the second level structure, stored sequentially in the leaf nodes. The approximate location $\hat{f}(t)$ at any normalized time instant $t \in [t_1, t_2]$ can be computed using Equation 1, as well as the spatial range $[\hat{f}(t) - \epsilon_k, \hat{f}(t) + \epsilon_k]$. If there is a time $t \in [t_1, t_2]$ such that the spatial computed compass is completely inside r, the trajectory segment is a filtering true hit, its ID will be reported. If this range does not intersect query r for any $t \in [t_1, t_2]$, the trajectory segment is pruned out. Otherwise, refinement is required for determining whether this trajectory segment is a true hit or false hit.

7 Experimental Evaluation

Since no real trajectory data sets are currently publicly available, we generated synthetic data sets using Brinkhof's network-based generator [2]. We used the TIGER data files for the road network in San Joaquin County, CA, and the road network in the city of Oldenburg, German. Our datasets were obtained by running the simulation for a total of 1000 timestamps. We focus mainly on the results of the datasets generated by the network-based generator, since it is has been extensively used in the previous work in this area [4,8,25]. Further, as indicated by some recent work [15,6], movement along roads has practical significance in real-world applications.

Datasets CA5k and OD5k have $5,000$ trajectories in all, and were generated with 6 object classes, 3 external object classes, $3,000$ initial objects, and 2 new objects per time-instant. We note that each object reports its position and movement function at every time instant during its lifetime, so the number of movement functions for each object will be the same as the duration of its lifetime. Table 1 shows the characteristics of our datasets.

We implemented the PA-tree with the Spatial Index Library of [7]. Our method is compared with the MVR-tree approach [9,8], which uses the LAGreedy algorithm



(a) Index construction. (b) Query evaluation.

Fig. 10. CPU cost

to model each trajectory with multiple MBRs. In the following figures, the legend `PA-tree` represents our method, while `MVR-tree` represents the method of [9,8].

Our experiments were run on an Intel Pentium IV 1.7Ghz processor, with 512 Mbytes of main memory. The page size is 4Kbyte in all experiments. We use a buffer with size being about 10% of the original dataset. Unlike [8], we do not reset the buffer before executing every query, since reseting the buffer will render the buffer useless when evaluating a workload of multiple queries. Further, we assume the ratio of cost of sequential I/O to that of random I/O is $1:20$ [5].

We use three types of query workloads, each containing 1000 queries with varying $qlen$, the length of temporal interval. The three workloads consist of queries with $qlen = 1$ for timestamp queries, $qlen = 50$ and $qlen = 100$ for medium and large time interval queries, respectively. Each query range is a rectangle uniformly distributed in the unit square, with the edge length being uniformly distributed in $[0, 0.1]$. In the following figures, the average query performances per query are reported.

We evaluated performance with respect to the size of index structures, by varying the number of MBRs for the MVR-tree or the number of interals for the PA-tree. For the MVR-tree, let $S\%$ represents $(1+S\%)N$ MBRs are used for a dataset of N trajectories. S is varied from 10 to 1000. For the PA-tree, we varied m, the number of intervals that the temporal domain is split into, from 5 to 50.

Clustered Index vs. Non-clustered Index. We tested the query performance for both clustered index and non-clustered index. For a clustered index, all the trajectory segments associated with the entries in a leaf node will be stored sequentially to that leaf node. For non-clustered index, same as the TB-tree [17], each data page consists of line segments belonging to the same trajectory. All the data page will be stored sequentially, according to the order of the start-time of the line segments (in case of a tier, trajectory id will be used), while each entry of leaf nodes will have a pointer to its data page.

Further, for clustered index, we notice that assigning all the available buffer to the index structure can reduce the overall I/O cost. This is because the data pages are sequentially retrieved, while the index pages are retrieved via random I/O. In contrast, for non-clustered index, both index pages and data pages could be random I/O. Therefore, we assign 50% buffer to the index structure, while 50% buffer to the data file.

7.1 Performance of Index Construction

For the MVR-tree, building the index structures involved assigning MBRs to each trajectory, creating MBRs for each trajectory, and loading the MBRs into MVR-trees. As pointed in [8], the first two steps are extremely expensive, since it requires one full database scan in order to compute the best approximation per trajectory. In contrast, building PA-trees is much more efficient, since each trajectory can be processed individually. We split each trajectory into segments according to the temporal domain splits, estimate the degree of polynomial approximation and insert the polynomial approximations into the PA-tree. Therefore, as Figure 10(a) shows, the cost of building the MVR-tree is about 50 times higher than that of building the PA-tree for the CA5k dataset. This clearly demonstrates the PA-tree will be a more appropriate choice when the dataset is extremely large, or when the trajectory data is collected at high rate, requiring on-line processing.

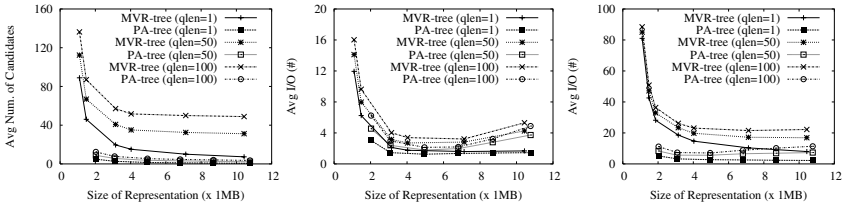
7.2 Query Performance

Executing query over the PA-tree requires us to compute the polynomials during the filtering step, so that the CPU cost will be higher than that of the MVR-tree. However, since the PA-tree has higher pruning power, we have a much smaller candidate set for the refinement step, so the CPU cost during the refinement step will be much smaller than the MVR-tree. As a result, in Figure 10(b), we can see that in most cases the overall CPU cost for the PA-tree is actually better than that of the MVR-tree. At any rate, the bottleneck is typically I/O, since CPU speeds tend to improve much faster than I/O speeds. We will therefore focus on the I/O cost, due to space limitations.

Figure 11 and 12 plots the IO performance for the dataset CA5k and OD5K, respectively. We only discuss CA5k in detail, since results for OD5k are quite similar. From Figure 11(a), we observe that both PA-tree and MVR-tree reduce the size of candidate set for the refinement step more effectively with larger index structures. This is expected, since increasing index size implies smaller dead space, and higher approximation quality results in fewer candidates. However, we can clearly see the PA-tree has significantly smaller candidate set than that of the MVR-tree, which is consistent with the comparison shown in Figure 5. Further, this disparity increases with $qlen$, since longer query period implies higher chance of false hits with MBR approximations.

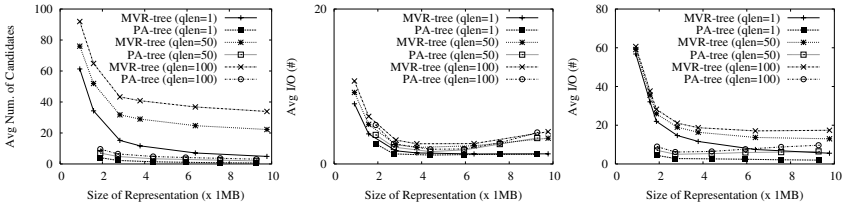
Performance with Clustered Indices. Figure 11(b) shows the total I/O cost including both filtering and refinement steps, in terms of the numbers of equivalent random I/O operations. For all types of workloads, the PA-tree incurs lower overall I/O cost than the MVR-tree. The improved approximation quality in the PA-tree requires checking of fewer index nodes and fewer candidates.

Figure 11(b) captures some interesting trade-offs. A larger index allows better pruning, lowering the number of candidates and I/O cost for the refinement step. However, since the buffer size is fixed, increasing the index size beyond a certain point causes the filtering-step I/O cost to overwhelm the benefits of better pruning. After that point, increasing index size yields no benefit. This results in an upward trend in the I/O cost, which is quite noticeable for the MVR-tree. This effect is stronger with clustered indices, for which a larger fraction of I/O costs are incurred in the filter step.



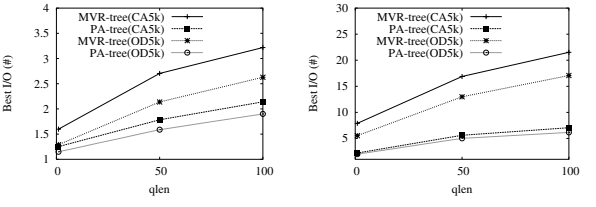
(a) Num. of candidates (b) IO (clustered) (c) IO (non-clustered)

Fig. 11. CA5k



(a) Num. of candidates (b) IO (clustered) (c) IO (non-clustered)

Fig. 12. OD5k



(a) clustered (b) non-clustered

Fig. 13. Best IO

For a given index size, the MVR-tree is able to split a trajectory into more segments than the PA-tree, since the PA-tree must hold more coefficients per segment. When a clustered index is used, only the line segments of the candidate segment, stored sequentially adjacent to the leaf nodes, will be retrieved. Therefore, the MVR-tree incurs lower I/O cost per candidate trajectory segment. (This advantage disappears for non-clustered indices, as we see shortly.) However, the PA-tree still requires lower refinement step costs than the MVR-tree due to its superior ability to reduce candidate set size.

Figure 13(a) plots the best I/O performance for MVR-tree and PR-tree over all possible index sizes. For clustered index, the MVR-tree is 20%–60% more expensive than the PA-tree.

Performance with Non-clustered Indices. Figure 11(c) plots the overall IO cost using non-clustered indices. PA-tree performance shows even greater improvements over the MVR-tree, and mirrors the improvements in candidate set size. For non-clustered

indices, a candidate requires at least one disk I/O, except for a buffer hit. Overall, as Figure 13(b) shows, the best I/O cost achieved with PR-tree is about 3–4 times lower than that for MVR-tree.

8 Conclusions and Future Work

In this paper, we have presented a new parametric indexing method suitable for large trajectory datasets, and for answering historical spatio-temporal queries efficiently. Our polynomial approximations method achieves much better performance than the general used MBR approximation. We present the PA-tree, a two-level structure for indexing trajectories using polynomial approximations. Our comprehensive experimental evaluations demonstrate that the PA-tree significantly outperforms current methods which uses MBR approximation, such as the MVR-tree. Consequently, the PA-tree is an extremely efficient and practical indexing structure for evaluating historical queries over trajectory data. As a future work, we are investigating the applicability of our methods to domains other than trajectory data, such as complex spatial objects.

Acknowledgments

This work was supported in part by grants from Tata Consultancy Services, Inc., the Digital Media Innovations Program of the University of California, and by award FTN F30602-01-2-0536 from the Defense Advanced Research Projects Agency.

References

1. M. Barth. UCR IntelliShare Project. `http://evwebsvr.cert.ucr.edu/intellishare/.`
2. T. Brinkhoff. Generating Network-Based Moving Objects. In *SSDBM'00*, page 253. IEEE Computer Society, 2000.
3. Y. Cai and R. Ng. Indexing Spatio-temporal Trajectories With Chebyshev Polynomials. In *SIGMOD Conference*, pages 599–610. ACM Press, 2004.
4. V. P. Chakka, A. Everspaugh, and J. M. Patel. Indexing Large Trajectory Data Sets With SETI. In *CIDR*, 2003.
5. L. Chung, B. Worthington, R. Horst, and J. Gray. Windows 200 Disk IO Performance. Microsoft technical report, MS-TR-2000-55, June 2000.
6. S. Gupta, S. Kopparty, and C. V. Ravishankar. Roads, Codes and Spatiotemporal Queries. In *PODS*, pages 115–124, 2004.
7. M. Hadjieleftheriou. Spatial Index Library. `http://www.cs.ucr.edu/ marioh/spatialindex/index.html.`
8. M. Hadjieleftheriou, G. Kollios, D. Gunopulos, and V. J. Tsotras. Indexing Spatio-temporal Archives. *The VLDB Journal*. to appear.
9. M. Hadjieleftheriou, G. Kollios, V. J. Tsotras, and D. Gunopulos. Efficient Indexing of Spatiotemporal Objects. In *EDBT '02*, pages 251–268. Springer-Verlag, 2002.
10. G. Kollios, D. Gunopulos, and V. J. Tsotras. On Indexing Mobile Objects. In *PODS '99*, pages 261–272. ACM Press, 1999.

11. G. Kollios, V. J. Tsotras, D. Gunopulos, A. Delis, and M. Hadjieleftheriou. Indexing Animated Objects Using Spatiotemporal Access Methods. *TKDE*, 13(5):758–777, 2001.
12. J. C. Mason and D. Handscomb. *Chebyshev Polynomials*. Chapman and Hall, 2003.
13. Federal Communications Commision. Enhanced 911. `http://www.fcc.gov/911/enhanced/`.
14. M. A. Nascimento and J. R. O. Silva. Towards Historical R-trees. In *Proceedings of the 1998 ACM symposium on Applied Computing*, pages 235–240. ACM Press, 1998.
15. D. Papadias, J. Zhang, N. Mamoulis, and Y. Tao. Query Processing in Spatial Network Databases. In *VLDB*, pages 802–813, 2003.
16. J. M. Patel, Y. Chen, and V. P. Chakka. STRIPES: An Efficient Index for Predicted Trajectories. In *SIGMOD Conference*, pages 637–646, 2004.
17. D. Pfoser, C. S. Jensen, and Y. Theodoridis. Novel Approaches in Query Processing for Moving Object Trajectories. In *VLDB'00*, pages 395–406. Morgan Kaufmann Publishers Inc., 2000.
18. K. Porkaew, I. Lazaridis, and S. Mehrotra. Querying Mobile Objects in Spatio-Temporal Databases. In *SSTD*, pages 59–78, 2001.
19. S. Saltenis, C. S. Jensen, S. T. Leutenegger, and M. A. Lopez. Indexing the Positions of Continuously Moving Objects. In *SIGMOD Conference*, pages 331–342, 2000.
20. Y. Tao, C. Faloutsos, D. Papadias, and B. Liu. Prediction and Indexing of Moving Objects with Unknown Motion Patterns. In *SIGMOD Conference*, pages 611–622, 2004.
21. Y. Tao and D. Papadias. MV3R-Tree: A Spatio-Temporal Access Method for Timestamp and Interval Queries. In *VLDB '01*, pages 431–440. Morgan Kaufmann Publishers Inc., 2001.
22. Y. Tao, D. Papadias, and J. Sun. The TPR*-Tree: An Optimized Spatio-Temporal Access Method for Predictive Queries. In *VLDB*, pages 790–801, 2003.
23. Y. Theodoridis, M. Vazirgiannis, and T. Sellis. Spatio-Temporal Indexing For Large Multimedia Applications. In *ICMCS '96*. IEEE Computer Society, 1996.
24. X. Xu, J. Han, and W. Lu. RT-tree: An Improved R-tree Index Structure For Spatiotemporal Databases. In *Proc. of the 4th Intl. Symposium on Spatial Data Handling*, 1990.
25. H. Zhu, J. Su, and O. H. Ibarra. Trajectory Queries and Octagons in Moving Object Databases. In *CIKM*, pages 413–421, 2002.