Primal or Dual: Which Promises Faster Spatiotemporal Search?

Yufei Tao Xiaokui Xiao

Department of Computer Science and Engineering
Chinese University of Hong Kong
New Territories, Hong Kong
{*taoyf, xkxiao*}*@cse.cuhk.edu.hk*

Abstract

The existing predictive spatiotemporal indexes can be classified into two categories, depending on whether they are based on the *primal* or *dual* methodology. Although we have gained considerable empirical knowledge about various access methods, currently there is only limited understanding on the theoretical characteristics of the two methodologies. In fact, the experimental results in different papers even contradict each other, regarding the relative superiority of the primal and dual techniques.

This paper presents a careful study on the query performance of general primal and dual indexes, and reveals important insight into the behavior of each technique. In particular, we mathematically establish the conditions that determine the superiority of each methodology, and provide rigorous justification for well-known observations that have not been properly explained in the literature. Our analytical findings also resolve the contradiction in the experiments of previous work.

To appear in VLDB Journal.
Keywords: Spatial Database, Range Search, Moving Objects, Theory

1 Introduction

A *predictive spatiotemporal database* maintains the current motion parameters of linearly moving objects, and supports efficient retrieval of objects whose expected future locations satisfy certain predicates. Such databases have been extensively studied in the past decade, since they play an imperative role in many applications including transportation monitoring, flight control, location-based services, etc.

A fundamental operation in these applications is *range search*, which specifies a (usually, rectangular) region Q, a future timestamp t, and reports all the objects whose locations at t appear in Q. For example, a query may retrieve "all the aircrafts expected to appear in the airspace of Washington 10 minutes later", where the spatial region is "Washington", and the query time "10 minutes from now".

We are interested in access methods that minimize the cost of range search. Obviously, the problem of indexing is much simpler, if all objects are stationary. In this case, a spatiotemporal database degenerates into a conventional, thoroughly studied, spatial database. There exist a large number of efficient spatial indexes, most notably, the R-tree [2].

Moving objects can be regarded as static points, as long as a single timestamp is concerned. Therefore, a natural idea of spatiotemporal indexing is to build a spatial access method at every future timestamp, capturing the "snapshot" of the dataset at that time. This is the motivation behind an important technique [17], which we refer to as the *primal method*. Informally (see the next section for details), as time progresses, the method allows an R-tree to evolve with time *automatically*, namely, no physical modification is necessary unless an object alters its motion parameters. A range query can be processed directly using the spatial range search algorithm, but on the evolved version of the index at the query time.

The *dual method* [12] is an alternative technique of spatiotemporal indexing. It applies a transformation that maps each moving object to a *stationary* point in a "dual space". Accordingly, for any range query Q, it is possible to construct another search region Q' in the dual space (as discussed in Section 2.1), such that if an object qualifies Q, its dual must fall in Q'. As a result, spatiotemporal range search is reduced to a purely spatial problem: retrieval of static points (i.e., duals) covered in a given area.

The spatiotemporal community has deployed numerous primal and dual indexes, as summarized in Section 2. After we have gained considerable empirical knowledge, it is time to address a deeper question: *which technique is better in theory?* We cannot (neither now nor in the future) answer the question by simply

checking which technique is taken by the current state-of-the-art access method. For example, the fact that the state-of-the-art belongs to the primal family does not imply that the primal approach is superior, because an "ideal" dual index may not have been discovered (and vice versa).

Contributions. This paper does not propose yet another index structure. Instead, we carry out a systematic theoretical study on the intrinsic properties that are possessed by all the (existing and future) primal and dual access methods, thus enhancing the understanding of the primal and dual methodologies in general. Our primary results can be summarized as follows.

First, for both primal and dual indexes, we establish a lower bound $\Omega(\sqrt{n})$ for the expected I/O cost of range search on uniform data, where n is the number of leaf nodes. Our analysis settles the hidden constant in the complexity, and thus, quantifies the smallest *actual* overhead in practice. Unfortunately, the same lower bound also applies to most, if not all, well-studied operations on moving objects, e.g., nearest neighbor search. This is a somewhat surprising fact, because a nearest neighbor query is typically highly efficient on stationary points (for realistic datasets, it can be solved in a very small number of I/Os [4], using an R-tree).

Second, we prove that, regardless of the dataset properties, the query cost of a dual index is "stable", since it is always limited by a certain upper bound. However, except at its construction time, a dual index never achieves the theoretical cost lower bound. On the other hand, deploying a primal access method is more "risky", in the sense that query efficiency of the index may continuously deteriorate with time, and eventually, become extremely poor. The advantage of a primal index is that, when certain conditions are satisfied, it may achieve the cost lower bound at all times, i.e., offering the optimal search performance.

Third, we show that the *dataset agility* (describing how many objects are updated at a timestamp, as will be formalized later) determines the relative query performance of primal and dual indexes. Specifically, when the agility exceeds a threshold, a primal index outperforms a dual access method. Otherwise (the agility is below the threshold), as time evolves, a dual method eventually entails smaller overhead. We present detailed derivation of the threshold, thus providing reliable guidance for choosing a suitable index in practice.

Finally, we apply our analytical findings to resolve the contradiction among the experiment results reported in the previous work [11, 16, 18, 22]. Specifically, primal indexes outperform dual access methods in [18, 22], whereas the opposite is argued in [11, 16]. This divergence causes confusion in the spatiotemporal community, and prevents practitioners from selecting the most appropriate technique. We show that all the results are correct, which seem inconsistent simply because they were obtained by setting several crucial

parameters in different ways, favoring the technique being championed.

The rest of the paper is organized as follows. Section 2 reviews the primal and dual methodologies. Section 3 formally defines the problem studied in this paper. Section 4 analyzes the cost lower bound of spatiotemporal search, while Section 5 investigates the different behavior of primal and dual indexes, respectively. Section 6 discusses update performance. Section 7 contains numerical and empirical results. Finally, Section 8 concludes the paper by summarizing the practical influence of our findings.

2 Primal and Dual Techniques

Saltenis et al. [17] propose the primal method by designing the TPR-tree. This structure has an enhanced version called the TPR*-tree [18]. The dual technique, on the other hand, is initiated by Kollios et al. [12], and improved in [1, 6, 13]. The dual category also includes STRIPES [16], the B^x-tree[1] [11], and the B^{dual}-tree [22].

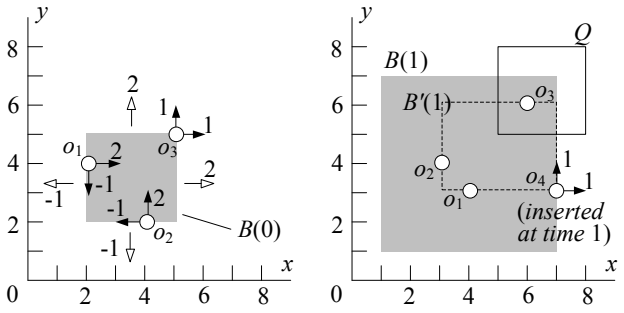
Assuming the knowledge of R-trees [2], here we provide an introduction to the primal and dual approaches. Our discussion proceeds in two steps. In Section 2.1, we explain the underlying rationales behind the two techniques, and elaborate their common properties. Then, Section 2.2 reveals the crucial difference between primal and dual indexes that determine their unique characteristics.

2.1 Equivalent Rationales

Primal. Figure 1a shows three 2D objects o_1, o_2, o_3 at the current time 0. For example, o_1 is at the coordinates $(2, 4)$, and moving with velocities 2 and -1 on the x- and y-dimensions respectively (we use black arrows to illustrate object velocities). A negative velocity means that the movement is towards the negative direction of an axis.

A primal access method can be regarded as an adapted R-tree indexing objects' locations. The most important adaptation is to augment the spatial bounding rectangle (SBR) of each node with velocities. Consider a leaf node containing o_1, o_2, and o_3, whose SBR is the grey rectangle $B(0)$ in Figure 1a, tightly enclosing the locations of the 3 objects. The node is associated with 4 velocities, which describe the movement of the edges of its SBR. Specifically, the velocity of the left/right edge equals the smallest/largest velocity of the objects in the node on the x-dimension (e.g., the velocities -1, 2 of the left and right edges are decided by

[1]The B^x-tree spans the boundary of the primal and dual techniques, since it utilizes features of both methodologies.



(a) Constructed at time 0 (b) Expanded at time 1

Figure 1: An expanding bounding rectangle

o_2 and o_1, respectively). Likewise, the velocities of the bottom and top edges define the range of y-velocities of the objects. Figure 1a demonstrates edge velocities with white arrows. The SBR and velocities of a node are stored in its parent.

The velocities of a node allow its SBR to expand over time, such that the SBR at any future time t always covers the locations of the objects (in the node) at time t. We illustrate this using a range query that requests the objects in rectangle Q (see Figure 1b) at future time 1 (the current time is 0). The grey box in Figure 1b shows the expanded SBR, denoted as $B(1)$, of the node in Figure 1a (e.g., the right edge of the SBR has moved 2 units). $B(1)$ intersects Q, and hence, the node must be visited to prevent false misses. Figure 1b also demonstrates objects' expected locations at time 1; o_3 qualifies the query (o_4 and the dashed rectangle will be discussed later).

An expanded SBR at any future time t is not physically stored, but it is computed dynamically during query execution from the node's SBR at time 0 and its velocities. In general, all the expanded SBRs at time t simulate a conventional R-tree managing the object locations at time t, except that an SBR is not necessarily tight. For example, in Figure 1, although $B(0)$ is the minimum SBR of o_1, o_2, o_3, $B(1)$ is larger than the minimum SBR of these objects at time 1.

Dual. The dual method does not generate a time-evolving index, but directly applies a stationary structure. For this purpose, a 2D moving point is mapped into a 4D dual space. For example, the dual of object o_1 in Figure 1a is $(2, 4, -1, 2)$, where the first (or last) two numbers indicate the object's coordinates (or velocities) on the x- and y-dimensions, respectively. This transformation is based on a *reference time* 0, since $(2, 4)$ is the location of o_1 at that time. In general, all duals must be calculated using the same reference time.

Given a range query, the dual technique transforms it into a "simplex region" in the dual space. Since it is

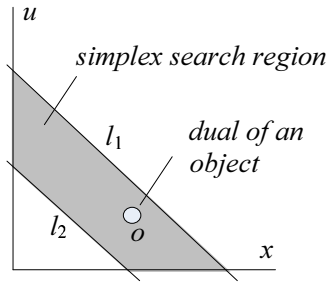


Figure 2: Dual transformation

not possible to visualize a 4D region, we illustrate the idea using 1D objects. In this case, the dual space has two dimensions, capturing an object's location x at the reference time 0, and velocity u, respectively.

Let $[c_1, c_2]$ be the search region (a 1D rectangle) of a range query, and t the query time. Obviously, if an object qualifies the query, it holds that $c_1 \leq x + u \cdot t \leq c_2$. This inequality defines the shaded area (a simplex region) in Figure 2, where line l_1 is described by $u = -\frac{1}{t}x + \frac{1}{t}c_2$, and line l_2 by $u = -\frac{1}{t}x + \frac{1}{t}c_1$. Retrieval of qualifying objects is reduced to finding all the duals in the shaded area (such as o in Figure 2).

Following the same idea, for 2D objects, a range query is transformed into a simplex region in the 4D dual space bounded by four linear hyper-planes. Discovery of object duals in such a region is a well-studied problem. A nice practical solution [7] is to create a 4D R-tree on the duals. To answer a query, a node needs to be visited, if and only if its 4D minimum bounding rectangle (MBR) intersects the simplex region.

Equivalence. Although presented from different perspectives, the rationales of the primal and dual techniques can be naturally bridged.

First, *a node in a primal index can be regarded as a 4D MBR in the dual space*. For example, the leaf node in Figure 1a defines a 4D MBR, whose projection on the two spatial dimensions (of the dual space) is $[2, 5]$, and its projection on the two velocity dimensions is $[-1, 2]$. As time progresses, the MBR is always minimum (in the dual space), since it keeps tightly enclosing the duals of the objects in the node (duals never move). By the same reasoning, *a dual node, represented as a 4D MBR in the dual space, can also be regarded as an expanding SBR in the primal space*.

In the sequel, we will use the expanding-2D-SBR and stationary-4D-MBR representations of a node interchangeably (no matter the node is from a primal or dual index). As an immediate corollary, in processing a range query, the node access conditions are identical for both types of indexes. Specifically, given a node in

a primal index, its expanded SBR intersects the query region at the query time, if and only if its 4D MBR in the dual space intersects the transformed simplex region. The reverse is true with respect to a node in a dual access method.

2.2 Different Clustering Effects over Time

A dual space must always be accompanied by a reference time, since there is a different dual space at every timestamp. For example, as discussed earlier, the dual of o_1 in Figure 1a is $(2, 4, -1, 2)$ at time 0. If the reference time equals 1, the dual of o_1 becomes $(1, 6, -1, 2)$.

As time evolves, the primal and dual techniques organize objects in different dual spaces. Specifically, a primal index always aims at clustering objects in the dual space *at the current time*, whereas a dual index performs clustering in the dual space *at a fixed historical timestamp*.

Next, we explain the above difference by elaborating the update strategy of each technique, focusing on insertion (deletion is relatively easy, since it involves simply finding an object and then removing it).

Primal. Assume a primal index constructed at time 0. A timestamp later, we need to insert an object o. For this purpose, the primal technique examines the 4D MBRs of the leaf nodes *in the dual space at the current time* 1. In Figure 1b, for instance, the 4D MBR of the node has projection $[1, 7]$ on the x- and y-dimensions, and projection $[-1, 2]$ on the two velocity dimensions. The leaf incorporating o is selected to minimize a certain quality metric. This is a leaf whose 4D MBR covers the dual of o at time 1. If no such MBR exists, the selected leaf is the one whose 4D MBR needs the "smallest" expansion to enclose o, where the degree of expansion is measured based on the quality metric.

Whenever a node is modified, the primal method performs *SBR tightening*, which shrinks the SBR of the node to minimum at the current time. For example, if object o_4 is added to the node in Figure 1b, the SBR of the node will be shrunk to the smallest rectangle (the dashed box) covering o_1, ..., o_4 at time 1. Tightening usually incurs no extra I/O because the SBR and velocities of the node may need to be updated in its parent anyway, in order to capture the newly inserted object (this is why tightening is carried out only if the node is modified). Furthermore, tightening the SBR also makes the 4D MBR of the node (in the dual space at the current time) smaller.

Each node is associated with a *reference time*, equal to the most recent timestamp when the node's SBR was tightened. In Figure 1a, the node has reference time 0. After the SBR-tightening at time 1, the node's

reference time changes to 1, so that an expanded SBR at a future timestamp can be calculated based on the SBR at time 1. Obviously, different nodes can have various reference times[2].

Dual. Regardless of the current time, the dual method always converts data into the dual space at the index construction time. Again, assume a dual index at time 0, and that we want to insert an object o at time $t > 0$. The dual approach converts o to its "counterpart" o' at time 0. Specifically, let x, y (or u, v) be the x-, y-coordinates (or -velocities) of o at time t. Then, o' shares the same velocities as o, but has coordinates $(x - u \cdot t,\ y - v \cdot t)$, i.e., the location of o at time 0. We incorporate the dual of o' in the index, which completes the insertion of o.

Hence, the reference time is a property of a dual index, common to *all* nodes. Updating the index is reduced to modifying a conventional spatial structure. Hence, objects are always well-clustered, but in the dual space at time 0.

As will be proved in Section 5.2, fixing the reference time to 0 has a serious drawback: query cost continuously grows with time (a primal index may not have this problem, which will be explained in Section 5.1). To solve the problem, we should periodically replace the old index with a new one built at a more recent timestamp. To achieve this with the same update overhead (as maintaining a single index), the dual technique adopts a *two-structure mechanism*, which places a requirement on objects: they must issue at least one update every T timestamps, where T is a system parameter. (This requirement may not necessarily be satisfied in practice, in which case some objects may be forced to generate additional updates. We will re-visit this issue later in Section 6.)

Specifically, the mechanism works as follows. At the initial time 0, a dual index $Index_1$ is built with reference time 0, and the other structure $Index_2$ is inactive. During the period $[0, T)$, all insertions/deletions are performed on $Index_1$. At time T, $Index_2$ is initialized with reference time T. During $[T, 2T)$, objects are inserted only in $Index_2$, but deletion may be performed on $Index_1$ or $Index_2$, depending on whether the corresponding object was inserted before or after T.

[2]The reference time of a node can be captured without spending any physical storage, using a trick proposed in [17]. Consider a node with reference time t. Assume that, along a dimension, the SBR of the node has extent $[x, x']$, and its VBR has extent $[v, v']$. Then, the parent entry of the node only needs to keep 4 values: y, y', v, v', where $y = x - v \cdot t$ and $y' = x' - v \cdot t$. These 4 values are sufficient to obtain the projection of the node's SBR on the corresponding dimension, at any future timestamp t_c. Specifically, the projection is $[z, z']$, where $z = y + v \cdot t_c$ and $z' = y' + v \cdot t_c$.

At time $2T$, $Index_1$ is definitely empty, because all objects inserted during $[0, T)$ must have been deleted. Hence, $Index_1$ is destroyed, and re-initiated with reference time $2T$. During $[2T, 3T)$, the roles of $Index_1$ and $Index_2$ are reversed. Namely, insertions are carried out in $Index_1$ only, but objects may be deleted from either tree, depending on their insertion time. Similarly, at time $3T$, $Index_2$ is empty, and re-initiated with reference time $3T$, after which the roles of the two structures are reversed again. The above process is repeated every T timestamps.

3 Problem Formulation

Unless specifically stated, our analysis focuses on 2D objects, since all the results can be extended to arbitrary dimensionalities in a straightforward manner. Without loss of generality, we assume that the spatial domain has a unit range $[0, 1]$ on the x- and y-dimensions, respectively. Along each dimension, the velocity of an object distributes in $[-V, V]$. In other words, a dual space has 4 axes: two *spatial dimensions* with range $[0, 1]$, and two *velocity dimensions* with range $[-V, V]$.

At timestamp 0, every object generates an insertion to register its initial location and velocities. Whenever its velocity changes, it issues an update, including a deletion followed by an insertion. In particular, the deletion removes the database tuple corresponding to the object's old velocities, while the insertion adds a tuple capturing its new velocity. Thus, the dataset cardinality N remains fixed at all times.

We adopt a *deletions-first* strategy. Specifically, at each timestamp (other than the initial timestamp 0), the database first collects objects' update requests at this timestamp. Then, all the deletions are processed first, before insertions are handled.

An object does *not* necessarily issue an update at every timestamp. We consider that every object has an equal probability of issuing updates. Hence, the number of updates at each timestamp accounts for a fixed percentage of the cardinality. We refer to the percentage as the *agility* of the dataset, and denote it as A. For example, a dataset with agility 0 contains objects that move with their initial velocities forever, while, in a dataset with agility 1, all the objects change velocities at every timestamp.

Construction of a primal/dual index requires an optimization goal. Following the previous work [17, 18], we aim at minimizing the average cost of "point queries", whose query time distributes uniformly in $[t_c, t_c + H]$, where t_c is the current time, and H a parameter called *horizon*. In particular, H is greater than 0, and controls how far into the future the index is optimized for. A *point query* is a special range query whose search region

is a point, i.e., a degenerated rectangle. Formally, if we denote $Q(t)$ as the expected cost of a query whose query time equals t, the cost metric of the index equals

$$CM(t_c) = \frac{1}{H} \int_{t_c}^{t_c+H} Q(t)\mathbf{d}t. \quad (1)$$

The quality of an index is *better* at the current time t_c, if its $CM(t_c)$ is *lower*. Note that the quality is a function of t_c.

We define the quality using point queries because they lead to the simplest formulae. In the same way, it is straightforward to formulate the quality using query regions with non-zero extents. All our analysis still applies, except that the resulting equations are more complex. As will be clear in Section 4.2, focusing on point queries allows us to explain the performance of other types of spatiotemporal queries as well.

We measure query cost as the number of *leaf nodes* accessed, which, in general, is significantly larger than the number of node accesses at the intermediate levels. This is especially true if a memory buffer is used; in that case, all the non-leaf levels may be retained in the buffer, so that accesses to those levels incur no I/O operations. Furthermore, all the existing spatiotemporal indexes have the same representation for leaf entries (each entry must store all the details of an object), even though they differ significantly at the intermediate levels.

Our derivation focuses on the *uniform data distribution*: at every timestamp, each coordinate and velocity of an object uniformly distribute in $[0, 1]$ and $[-V, V]$, respectively. The reasons for discussing uniform data are three-fold. First, this is a popular distribution experimented by the work [11, 16, 18] proposing recent spatiotemporal indexes. Second, we must make certain simplifying assumptions to allow rigorous probabilistic analysis, as in the existing studies on R-tree performance [20]. The uniform assumption minimizes the complication caused by data properties, making it easier to discover the intrinsic characteristics of each indexing technique. The third reason is that, a real data distribution can often be approximated as piece-wise uniform, as confirmed by the success of bucket-based histograms [5, 20, 19] in selectivity/query-cost estimation. When this is true, we can divide the dataset into several parts, such that the distribution within each part is close to uniformity. Then, we apply the observations from uniform-analysis "locally" to each part, for explaining the behavior of the index on the objects there. This approach has been applied in R-tree cost analysis [20].

Table 1 summarizes the set of symbols that will be used frequently in the subsequent analysis. Some symbols have not appeared so far, and will be introduced later.

Symbol	Description
t_c	the current time
H	the horizon of a primal index
T	the length of an update period of a dual index
$CM(t)$	the quality of an index at time t
V	the maximum absolute velocity value
n	the number of leaf nodes
N	the dataset cardinality
A	the dataset agility
f	the average node fanout
s	the spatial extent of a leaf MBR
w	the velocity extent of a leaf MBR

Table 1: Frequently used symbols

4 Basic Results

In this section, we study the quality (Equation 1) of primal and dual indexes *right after they are constructed*. The results are fundamental to studying the behavior of alternative structures as time evolves, which is the topic of Section 5.

The following analysis applies to both primal and dual indexes, which have the same query performance at the construction time 0, due to their equivalence discussed in Section 2.1. Particularly, a node in a primal/dual index can be regarded as an expanding 2D SBR in the primal space, or alternatively, a stationary 4D MBR in the dual space at time 0. Throughout this paper, we omit '2D' and '4D', since all SBRs and MBRs will be two- and four-dimensional respectively, unless otherwise stated. For the same reason, in this section, we ignore "at time 0" when referring to the dual space.

4.1 Lower Bound of Index Quality

Let us focus on a specific leaf node. Denote s as the projection length of its MBR on the two spatial dimensions (for uniform data, the projections on both dimensions are equally long). Similarly, we use w to represent the projection length of the MBR on the velocity dimensions.

Given a point query q at time $t \geq 0$, the node is visited, if and only if its expanded SBR at time t covers q. In particular, the SBR is a 2D square with extent length $s + w \cdot t$. When q randomly distributes in the spatial domain (with area 1), the probability that the node is accessed equals $(s + w \cdot t)^2$. Note that this probability

holds, regardless of the *position* of the node's MBR in the dual space[3] (i.e., the probability depends on only the *size* of the MBR).

Let n be the number of leaf nodes. In practice, n equals N/f, where N is the dataset cardinality, and f the average number of entries per node (f is a system parameter determined by the page size). Since data characteristics are identical throughout the dual space, the MBRs of all the leaf nodes have the same sizes, i.e., s and w on the spatial and velocity dimensions, respectively. Therefore:

$$Q(t) = n(s + w \cdot t)^2 \tag{2}$$

which gives the expected cost of a point query at time t.

To obtain the index quality at time 0, we plug the above formula into Equation 1 (replacing t_c with 0) which results in (after solving the remaining integral with respect to t):

$$CM(0) \quad = \quad n\left(s^2 + s \cdot w \cdot H + w^2 H^2/3\right) \tag{3}$$

The dual of each object must be enclosed in at least one leaf MBR. For a large dataset, the data density is high in the dual space, in which case we can consider that the union of all MBRs covers the entire space. Under such circumstances and based on the fact that $CM(0)$ is monotonic with s and w, $CM(0)$ is minimized when all MBRs form a regular *tiling* of the dual space, i.e., no overlap among the MBRs. Specifically, the tiling is a 4D matrix of 4D rectangles, such that there are $1/s$ rectangles on each spatial dimension of the matrix, and $2V/w$ on a velocity dimension (recall that a velocity ranges in an interval $[-V, V]$ with length $2V$). Since the total number of rectangles equals n, we have

$$(1/s)^2 \cdot (2V/w)^2 = n \tag{4}$$

The above equation is valid only when $s \ll 1$ and $w \ll 2V$, i.e., the leaf MBRs are adequately partitioned along all dimensions of the dual space. This is true when the dataset cardinality is large, and H is meaningful, i.e., H is neither very small nor very large. In particular, an excessively small (or large) H leads to an index optimized for queries at the current time (or a long future period), rendering leaf partitioning to be performed only on the spatial (or velocity) dimensions.

Subject to the constraint of Equation 4, Equation 3 is minimized when

$$s^2 = w^2 H^2/3 \tag{5}$$

[3]Here, we tackle the "boundary effect", using the "wrapping model" commonly assumed in the spatial literature [15, 20].

From Equations 4 and 5, we obtain the s and w that produce the best index quality:

$$s = \left(\sqrt{2V \cdot H}\right) / (3n)^{0.25} \quad (6)$$

$$w = (12/n)^{0.25} \sqrt{V/H} \quad (7)$$

Substituting s and w of Equation 3 with their values in Equations 6 and 7 respectively, we obtain the lower bound for $CM(0)$:

$$2(2/\sqrt{3} + 1) \cdot V \cdot H \cdot \sqrt{n} \quad (8)$$

The above formula quantifies the quality of a primal/dual index (at time 0) in the *best case*, achievable by a "sufficiently good" implementation. In particular, the implementation should be able to create leaf nodes with flexible MBR sizes. As shown in Equations 6 and 7, the best s and w rely on numerous factors, which discourages using a structure like the quad-tree as the basis of a spatiotemporal access method. Specifically, a quad-tree creates leaf nodes by recursively breaking a quadrant into 4 equal-sized sub-quadrants (using the dual space as the original quadrant). As a result, the MBRs of leaf nodes have rigid side lengths, which are limited to $1/2^i$ of the lengths of the corresponding dimensions, for some integer i.

We point out that Equations 6 and 7 also confirm a relevant result in [17], which shows that the best ratio s/w (for minimizing query cost) equals $H/\sqrt{3}$.

4.2 "$\sqrt{n}$ Nature" of Spatiotemporal Queries

Formula 8 shows that the average cost of point search is at least at the order of $\sqrt{n}$ (the formula has not included the overhead of accessing the non-leaf levels). For comparison, here we mention a well-known result on spatial databases [20]: given a set of stationary points that uniformly distribute in 2D space, an R-tree permits processing any point search by accessing only 1 leaf node in expectation!

The comparison reveals a pessimistic fact: *many types of queries that used to be cheap in the spatial scenario become expensive on moving objects*. A typical example is *nearest neighbor* (NN) *search* [9], which finds the data point closest to a query point q. The definition naturally extends to spatiotemporal databases [3], where a NN query specifies an additional timestamp.

For both stationary and moving data in 2D space, the cost of a NN query at q is analogous to that of performing point search at q (obviously, for moving data, the NN and point queries should share the same query timestamp). This is a property of the "best-first" algorithm [9], which is the state-of-the-art for NN search

in low-dimensional spaces. In spatial databases, a NN query is very efficient, and usually terminates by accessing a single path of an R-tree. In spatiotemporal environments, however, its cost is significantly higher: $\Omega(\sqrt{n})$ leaf nodes are expected to be visited.

In fact, Formula 8 implies a more general result: *all spatiotemporal queries, which are provably more expensive than point search, have cost complexity* $\Omega(\sqrt{n})$ *in expectation*. Unfortunately, such queries include (i) the spatiotemporal counterparts of all well-studied spatial operations, e.g., reverse NN search [3], aggregate retrieval, etc., and (ii) operations specific to moving objects such as continuous retrieval [3], location-based queries [10], and so on. Furthermore, although our analysis so far considers only timestamp 0, the situation is actually worse at subsequent timestamps, as will be clarified in the next section.

Kollios et al. [12] also establish a cost lower bound $O(\sqrt{n})$ for the type of queries targeted by our analysis. Care is needed to interpret the two bounds. The one by Kollios et al. applies to the most-adversely designed *one-dimensional* datasets and queries. In other words, their bound implies that one cannot hope to design a data structure that consumes linear space, and answers any (1D) query on any (1D) dataset in less than $O(\sqrt{n})$ I/Os. Kollios et al. [12] in fact develop another cost lower bound $O(n^{3/4})$ for the 2D case. Our lower bound $O(\sqrt{n})$, on the other hand, concerns the average performance of a *special* type of queries on a *special* type of 2D datasets, i.e., *uniform* queries on *uniform* data. These bounds do not contradict each other. Specifically, in the 2D space, even if $O(\sqrt{n})$ expected cost might be achievable by some index structure (occupying $O(n)$ space) for uniform queries issued on uniform data, that structure necessarily entails at least $O(n^{3/4})$ I/Os on the worst dataset and query.

5 Advanced Results

The previous section focused on t_c (the current time) = 0 (index construction time). We proceed to discuss how the index quality $CM(t_c)$ (Equation 1) changes with t_c. We will first study primal indexes in Section 5.1, before analyzing dual solutions in Section 5.2. Finally, Section 5.3 compares the characteristics of the two techniques, and identifies the better technique in different scenarios.

5.1 Primal

If the dataset agility is exceedingly low (i.e., very few object updates), the quality of a primal index keeps deteriorating with time. As an extreme case, if the agility equals 0 (all objects move at their initial velocities permanently), SBR tightening (as explained in Section 2.2) is never performed, rendering leaf SBRs to grow

increasingly larger as time evolves, which in turn leads to higher query overhead.

On the other hand, if the agility is sufficiently high, the index will remain equally efficient. This is most obvious when the agility is 1, i.e., all the objects are updated at each timestamp. In this case, the entire index is destroyed at each timestamp, and then re-built, thus trivially retaining the maximum quality (Formula 8).

The above facts suggest a "magic threshold", such that when the agility reaches the threshold, a primal index offers the optimal query performance at all times! Next, we confirm this phenomenon with theoretical justification, and quantify the threshold.

5.1.1 Why Would Quality Deteriorate?

This subsection considers the following problem. Assume that, at time 0, we build a primal index with the optimal query performance, i.e., the extent lengths of the 4D leaf MBRs satisfy Equations 6 and 7. At time 1, $A \cdot N$ objects issue updates, where N is the dataset cardinality, and A the agility. What are the conditions to be satisfied, if the resulting index at time 1 offers the same (optimal) query efficiency?

The index quality at time 1 can also be represented by Equation 3 (replacing $CM(0)$ with $CM(1)$), except that s and w should be interpreted as the extent lengths of the 4D leaf MBRs in the dual space at time 1 (as opposed to time 0). Hence, if there is no deterioration (of query performance) at time 1, s and w should also satisfy Equations 6 and 7, respectively.

In processing the object updates at time 1, the extents of a leaf node undergo a series of changes. For uniform data distribution, the behavior of all leaves is analogous. From a probabilistic point of view, since the number n of leaf nodes is large, it is safe to ignore the minor discrepancies among different nodes. Hence, to facilitate our analysis, we consider that all leaves behave in the same manner. As a corollary, $A \cdot N/n = A \cdot f$ objects are deleted and inserted in each leaf node, where f is the average node fanout.

Let us denote $B(0)$ and $B(1)$ as the 2D SBR of any leaf node at timestamps 0 and 1, respectively. Similarly, we use $M(0)$ and $M(1)$ to represent the 4D MBRs of the node in the dual spaces at time 0 and 1, respectively. In other words, $B(0)$ (or $B(1)$) is the projection of $M(0)$ (or $M(1)$) onto the spatial dimensions.

It suffices to discuss only the case where $A \cdot f \geq 1$, that is, *each leaf node receives at least an update at time* 1. Otherwise (a node is not updated), no SBR tightening is performed for this node, whose SBR at time 1 is thus expanded from its SBR at time 0 (c.f. the grey boxes in Figures 1a and 1b). Hence, the s (the SBR's

side length) at time 1 must be larger than its (optimal) value at time 0, resulting in quality degradation.

Under our deletions-first update strategy (deletions are processed before insertions; see Section 3), the extents of a node go through a *first-shrink-then-grow* process at time 1. Specifically, the extents keep shrinking during deletions, and then, are continuously enlarged in handling insertions. Next, we clarify the two phases in turn.

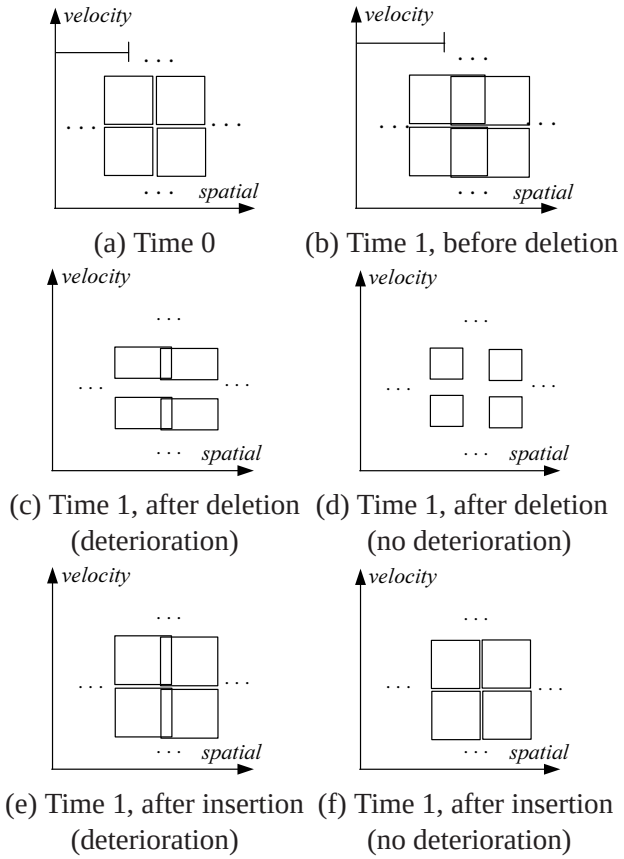
The Shrinking Phase. At time 1, before any deletion is performed, the node extents are the largest. We denote $B_{max}(1)$ and $M_{max}(1)$ as the SBR and MBR of the node at this moment. Compared to $M(0)$, $M_{max}(1)$ is longer only on the spatial dimensions, i.e., $M(0)$ and $M_{max}(1)$ have identical extents on the velocity dimensions. In particular, the spatial projection $B_{max}(1)$ of $M_{max}(1)$ is the expanded version (at time 1) of the spatial projection $B(0)$ of $M(0)$, based on the velocities of $B(0)$.

As objects are deleted from the leaf node, the SBR of the node shrinks (from $B_{max}(1)$), driven by two factors. First, after the first deletion, the SBR is immediately tightened. Furthermore, the SBR will remain tightened after the subsequent deletions (and also insertions in the growing phase). Second, as objects deciding the boundaries of the SBR are removed, the SBR becomes even smaller.

The velocity projection of the MBR, on the other hand, may also shrink as we perform deletions. However, the shrinking is only because objects at the boundary of the projection disappear, similar to the second factor mentioned earlier for SBRs (recall that tightening affects only spatial dimensions). It follows that the SBR and MBR of the node are the smallest, at the moment when all deletions are finished. We use $B_{min}(1)$ and $M_{min}(1)$ to represent their extents at this moment, respectively.

Before the enlargement phase starts, the current leaf MBRs (after deletions) no longer cover the whole dual space. *They have created gaps along the velocity dimensions, but NOT necessarily so along the spatial dimensions.* Since we cannot visualize a 4D dual space, let us illustrate this on 1D moving objects (the idea extents to any dimensionality naturally), for which the dual space has one spatial and one velocity dimension.

Figure 3a shows the leaf MBRs after the index is constructed at time 0 (the MBRs form a regular tiling of the dual space). Figure 3b demonstrates the situation at time 1 before deleting any object, i.e., the moment when the MBR extents are the largest (corresponding to $M_{max}(1)$ in our earlier analysis). Note that the centroid of each MBR has moved on the spatial dimension from its position at time 0, as indicated by the lengths of the horizontal segments above the MBRs in Figures 3a and 3b, respectively. Note that, these MBRs overlap



(a) Time 0 (b) Time 1, before deletion

(c) Time 1, after deletion (deterioration) (d) Time 1, after deletion (no deterioration)

(e) Time 1, after insertion (deterioration) (f) Time 1, after insertion (no deterioration)

Figure 3: Transitions of MBRs in the dual space

on the spatial dimension, but remain disjoint on the velocity dimension.

After all deletions, the extents of an MBR are reduced along all dimensions. However, depending on the amount of shrinking on the spatial dimension, the resulting MBRs may end up with two situations, as shown in Figures 3c and 3d respectively (corresponding to $M_{min}(1)$). There are gaps between two consecutive rows of MBRs on the velocity dimension in both cases, which, however, differ in whether MBRs may overlap on the spatial dimension (i.e., MBRs shrink less in Figure 3c). As analyzed in Section 5.1.2, the amount of spatial-extent shrinking depends on several factors including, very importantly, the dataset agility.

The Growing Phase. Inserting an object means first computing its dual at the current time 1, and then including the dual into the leaf MBR whose enlargement (for covering the object) incurs the smallest penalty (various penalties are adopted in different primal indexes; e.g., the TPR-tree [17] applies the "integrated volume", while the TPR*-tree [18] uses the areas of "sweeping regions"). As data is inserted, the MBRs will become larger, in order to fill the gaps among them. After all insertions, the union of all MBRs will again

cover the entire dual space (for a large dataset, object duals appear everywhere in the space).

Figure 3e shows the final MBRs for the situation of Figure 3c. Since the spatial-extents of the MBRs are already overlapping in Figure 3c, during insertions, the MBRs will be enlarged only on the velocity dimension (i.e., no gap to fill along the spatial dimension). In this case, the spatial projection length s of an MBR exceeds the corresponding value at time 0, i.e., the index has degraded.

In Figure 3d, on the other hand, each MBR is smaller than the corresponding MBR at time 0 (Figure 3a) on all dimensions. Hence, when insertions are performed, an MBR can be shaped again to the optimal extents, as demonstrated in Figure 3f. In this case, the index retains the same query efficiency as at time 0.

From the above analysis, it is clear that two conditions should be fulfilled, if a primal index incurs no deterioration at time 1:

1. The agility A must be at least $1/f$, so that each leaf node can receive at least an update.
2. The side length of $B_{min}(1)$ (i.e., the SBR of a node after all deletions) must be smaller than the value of s in Equation 6, so that the MBR of the node can "bounce" back to its size at time 0. This condition actually also indicates that the agility A must be adequately high, since the higher A is, the shorter the extents of $B_{min}(1)$.

Finally, we point out that the above analysis also applies to subsequent timestamps. Specifically, given a primal index with no degradation at time 1, the same conditions should be satisfied for the index to retain efficiency at the next timestamp, too.

5.1.2 Computing the Deterioration Threshold

To obtain the deterioration threshold (i.e., the lowest agility validating Conditions 1 and 2 in Section 5.1.1), we need to find the smallest agility fulfilling the second condition (Condition 1 indicates a constant agility $1/f$). The analysis is the same for all leaf nodes, regardless of where their MBRs are in the dual space at time 0. Hence, to simplify notation, we consider a node whose MBR is cornered at the origin of the dual space. Furthermore, it suffices to discuss 1D objects, because the deterioration threshold is identical for data of any dimensionality.

Specifically, the problem is as follows. We have f 1D moving points at time 0. Their locations and velocities are uniformly distributed in ranges $[0, s]$ and $[0, w]$, respectively. At time 1, $A \cdot f$ points are randomly deleted.

For the remaining $(1-A)f$ points, we use l to represent the length of the minimum 1D SBR enclosing their locations at time 1. What is the lowest A such that the expected l is at most s (represented in Equation 6)?

Rationale. Let x_i be the location of the i-th ($1 \leq i \leq (1-A)f$) remaining point at time 0, and u_i its velocity (i.e., x_i and u_i are uniform in $[0, s]$ and $[0, w]$, respectively). Hence, if λ_i is the location of the point at time 1, we have $\lambda_i = x_i + u_i$. The value of l thus equals

$$l = \max_{i=1}^{(1-A)f} \lambda_i - \min_{i=1}^{(1-A)f} \lambda_i \tag{9}$$

Clearly, l distributes in range $[0, s+w]$. The rest of this subsection will be devoted to deriving the probability $P\{l \leq \epsilon\}$ that l is at most a particular value $\epsilon \in [0, s+w]$. This probability will be a function of A and ϵ.

Once $P\{l \leq \epsilon\}$ is available, by taking its derivative against ϵ, we obtain the probability density function $pdf(l = \epsilon)$, which is also a function of A and ϵ. Then, setting the expected l to s, we obtain an equation of A:

$$\int_0^{s+w} \epsilon \cdot pdf(\epsilon)\, \mathbf{d}\epsilon = s \tag{10}$$

The solution of A is thus the lowest agility satisfying Condition 2 of the previous section. The solution cannot be represented as a closed formula, but can be obtained numerically. In Section 7.3, we will list the solutions under various settings (e.g., different H, n, f, etc.).

Detailed Derivation. We first analyze the distribution of λ_1, λ_2, ..., $\lambda_{(1-A)f}$. Since these $(1-A)f$ random variables are symmetric, it is sufficient to focus on one of them. In the sequel, we drop the subscript of λ (and accordingly, also the subscripts of x and u), when there is no ambiguity.

Lemma 1. *The distribution of λ is described by*

$$pdf(\lambda) = \begin{cases} \lambda/(w \cdot s) & \text{if } \lambda \in [0, w) \\ 1/s & \text{if } \lambda \in [w, s) \\ \frac{s+w-\lambda}{w \cdot s} & \text{if } \lambda \in [s, s+w) \\ 0 & \text{otherwise} \end{cases} \tag{11}$$

where s and w are given in Equations 6 and 7, respectively.

Proof. Obviously, λ belongs to $[0, s+w]$. To obtain its pdf, we first compute the probability $P\{\lambda \leq \epsilon\}$ that λ does not exceed a particular value $\epsilon \in [0, s+w]$. Recall that $\lambda = x + u$; hence, $\lambda \leq \epsilon$ means $x \leq \epsilon - u$.

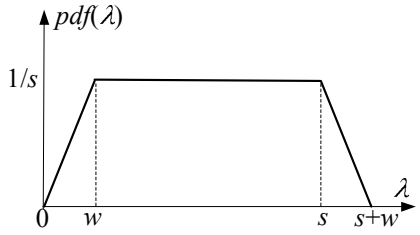


Figure 4: Visualization of $pdf(\lambda)$

As a result,

$$P\{\lambda \leq \epsilon\} \quad = \quad \frac{1}{w}\int_0^w P\{x \leq \epsilon - u\}\mathbf{d}u \tag{12}$$

If ϵ falls in $[0, w]$, Equation 12 becomes

$$P\{\lambda \leq \epsilon\} = \frac{1}{w \cdot s}\int_0^\epsilon (\epsilon - u)\mathbf{d}u = \frac{\epsilon^2}{2w \cdot s} \tag{13}$$

Taking the derivative of the above equation against ϵ, we obtain $pdf(\lambda = \epsilon) = \epsilon/(w \cdot s)$, thus proving the first case in the lemma ($\lambda \in [0, w]$). The other cases can be established similarly. □

Figure 4 provides the visualization of $pdf(\lambda)$, which is a trapezoidal shape, symmetric by the vertical line of $\lambda = (s + w)/2$. The above lemma assumes $w \leq s$. However, s/w always equals $H/\sqrt{3}$ (as is clear from Equation 4); hence, $w \leq s$ holds, as long as $H \geq \sqrt{3}$.

We are ready to calculate $P\{l \leq \epsilon\}$, where l is defined in Equation 9. For convenience, let us introduce

$$\alpha = \min_{i=1}^{(1-A)f} \lambda_i, \text{ and } \beta = \max_{i=1}^{(1-A)f} \lambda_i,$$

i.e., $l = \beta - \alpha$. Clearly, $0 \leq \alpha \leq \beta \leq s + w$. Hence, $P\{l \leq \epsilon\}$ = $P\{\beta - \alpha \leq \epsilon\}$, which is equivalent to

$$\int_0^{s+w} pdf(\alpha) \cdot P\{\beta - \alpha \leq \epsilon | \alpha\}\mathbf{d}\alpha$$

Thus, it remains to solve the two components of the above integral, i.e., $pdf(\alpha)$ and $P\{\beta - \alpha \leq \epsilon|\alpha\}$. We achieve this in two separate lemmas:

Lemma 2. *When* $\alpha \in [0, w)$, $pdf(\alpha) =$

$$\frac{\alpha \cdot (1 - A)f}{w \cdot s}\left(1 - \frac{\alpha^2}{2w \cdot s}\right)^{(1-A)f-1}$$

When $\alpha \in [w, s)$, $pdf(\alpha) =$

$$\frac{(1-A)f}{s}\left(1-\frac{2\alpha-w}{2s}\right)^{(1-A)f-1}$$

When $\alpha \in [s, s+w)$, $pdf(\alpha) =$

$$\frac{f(1-A)(s+w-\alpha)}{w\cdot s}\left(1-\frac{s(2\alpha-s)-(\alpha-w)^2}{2w\cdot s}\right)^{f(1-A)-1}$$

In any other case, $pdf(\alpha) = 0$.

Proof. The value of α is at most ϵ, if and only if one of λ_1, ..., $\lambda_{(1-A)f}$ is at most ϵ. Hence,

$$P\{\alpha \le \epsilon\} = 1 - (1 - P\{\lambda \le \epsilon\})^{f\cdot(1-A)}$$

where $P\{\lambda \le \epsilon\}$ is given by Equation 12. According to Equation 13, for $\alpha \in [0, w]$, we have

$$P\{\alpha \le \epsilon\} = 1 - \left(1 - \epsilon^2/(2w \cdot s)\right)^{(1-A)f}$$

Taking the derivative of the above equation against ϵ, we obtain $pdf(\alpha = \epsilon)$, which results in the first formula of the lemma. The other formulae can be established in the same way. □

Lemma 3. $P\{\beta - \alpha \le \epsilon | \alpha\}$ *equals*

$$\left(\frac{\int_{\alpha}^{\min\{\alpha+\epsilon, s+w\}} pdf(\lambda)\mathbf{d}\lambda}{\int_{\alpha}^{s+w} pdf(\lambda)\mathbf{d}\lambda}\right)^{(1-A)f-1} \tag{14}$$

where $pdf(\lambda)$ *is given in Lemma 1.*

Proof. The a priori condition "$|\alpha$" states that one of the $(1-A)f$ random variables λ_1, ..., $\lambda_{(1-A)f}$ must be α. Since these variables are symmetric and independent, without loss of generality, assume $\lambda_1 = \alpha$. In this case, each of λ_2, ..., $\lambda_{(1-A)f}$ must be in range $[\alpha, \min\{\alpha+\epsilon, s+w\}]$ (provided that it is at least α), the probability of which is captured by Equation 14. □

5.2 Dual

As mentioned in Section 2.2, a dual index adopts the two-structure mechanism to perform efficient periodic re-building. Before elaborating the behavior of the mechanism, we need to examine a fundamental issue: given a *single* dual index constructed at time 0, what is its best possible quality at timestamp $t \geq 0$? Equivalently, what is the lower bound of $CM(t)$ as in Equation 1? The following lemma provides the answer.

Lemma 4. *For a dual index with reference time 0, at any time $t \geq 0$, $CM(t)$ is at least $g(t) \cdot \sqrt{n}$, where* $g(t) =$

$$2V((2/\sqrt{3})\sqrt{3t^2 + 3t \cdot H + H^2} + 2t + H) \tag{15}$$

Proof. Let s (or w) be the spatial- (or velocity-) extent length of a leaf MBR. Plugging Equation 2 in Equation 1 and solving the remaining integral of t, we have $CM(t) =$

$$n(s^2 + s \cdot w(2t + H) + w^2/3 \cdot (H^2 + 3t \cdot H + 3t^2)) \tag{16}$$

Following the analysis in Section 4.1 for obtaining Equations 6 and 7, we obtain the s and w that minimize $CM(t)$:

$$\begin{aligned} s &= \sqrt{2V}(3t^2 + 3t \cdot H + H^2)^{0.25}/(3n)^{0.25} \\ w &= (12V^2/n)^{0.25}/(3t^2 + 3t \cdot H + H^2)^{0.25} \end{aligned}$$

Substituting s and w in Formula 16 with the above equations, we arrive at the representation of $CM(t)$ in the lemma. □

Lemma 4 essentially generalizes the results in Section 4, because, for $t = 0$, $g(t)\sqrt{n}$ degenerates into Formula 8. Furthermore, the lemma also confirms the necessity of the two-structure mechanism: if only a single structure (built at time 0) is deployed, the query cost will increase (roughly) linearly with time t.

Now, we turn our attention to a dual index with two structures $Index_1$ and $Index_2$, each of which becomes empty and is re-created every $2T$ timestamps (as described in Section 2.2). Hence, the behavior of the index demonstrates a periodic pattern, with $[T, 2T)$ being the first period, $[2T, 3T]$ the second, and so on. Note that the first period starts at T (instead of 0) because, during $[0, T)$, $Index_2$ is inactive. Furthermore, the length of a period is T (as opposed to $2T$), because the roles of the two indexes are switched every T timestamps.

Without loss of generality, let us focus on a single period $[T, 2T)$. We aim at representing the best possible index quality as a function of the current time $t_c \in [T, 2T)$. In particular, since a query must search both $Index_1$ and $Index_2$, the index quality equals

$$CM_1(t_c) + CM_2(t_c) \tag{17}$$

where, for $1 \leq i \leq 2$, $CM_i(t_c)$ is given in Equation 1, capturing the overhead of searching $Index_i$.

During $[T, 2T)$, $Index_1$ and $Index_2$ have reference times 0 and T, respectively. We denote $n_1(t_c)$ as the number of leaf nodes in $Index_1$ at time t_c, and similarly, $n_2(t_c)$ with respect to $Index_2$. At all times, the sum of $n_1(t_c)$ and $n_2(t_c)$ is a constant $n = N/f$, where N is the dataset cardinality, and f the average fanout. Furthermore, since $Index_1$ contains the entire dataset (or is empty) at time T (or $2T$), it holds that $n_1(T) = n$ and $n_1(2T) = 0$.

In general, there are $A \cdot N$ updates at every timestamp, where the agility A is at least $1/T$ (an object must have issued at least one update within T timestamps). Since each object removed from $Index_1$ is added to $Index_2$, $Index_1$ loses $A \cdot N$ objects (or $A \cdot n$ leaf nodes[4]) at each timestamp in the time interval $[T, T+1/A)$, and remains empty during $[T + 1/A, 2T)$. Therefore, for any $t_c \in [T, 2T]$, we have

$$n_1(t_c) = n\,(1 - A \cdot \min\{t_c - T, 1/A\}) \tag{18}$$

$$n_2(t_c) = n - n_1(t_c) \tag{19}$$

Computation of Formula 17 can be reduced to Lemma 4, but with care: the lemma assumes an index with reference time 0, whereas the reference time of $Index_2$ is T. In fact, Lemma 4 is always correct, as long as we interpret t as the difference between the current time and the reference time of the index. Hence, Formula 17 becomes

$$g(t_c) \cdot \sqrt{n_1(t_c)} + g(t_c - T) \cdot \sqrt{n_2(t_c)} \tag{20}$$

where function g is defined in Equation 15, $n_1(t_c)$ and $n_2(t_c)$ in Equations 18 and 19, respectively.

5.3 Which Technique is Faster in Theory?

There is no absolute answer, because each (primal or dual) technique has its unique characteristics, and has better query performance in some scenarios. The dataset agility A turns out to be the most important factor

[4] At each timestamp, the objects deleted are amortized among all the leaf nodes. These nodes may entail underflows, and hence, be merged into a smaller number of nodes after all deletions.

that determines the relative superiority of alternative solutions.

As shown in Section 5.1, a primal index offers the optimal query efficiency at all times, as long as A exceeds a threshold (depending on the solution of A from Equation 10). However, when this condition is violated, the index risks continuously deteriorating with time, such that eventually object clustering at the leaf level may become completely arbitrary, i.e., each leaf node contains f random objects (this phenomenon was also observed in [18]).

Regardless of the value of A, choosing a dual index is a "safe bet", because its quality changes periodically, and *never exceeds an upper bound*. This bound equals the maximum value of Formula 20, as t_c varies from T to $2T$ (i.e., within a period). Note that the t_c that produces the upper bound is not necessarily the center of a period, but varies depending on H, A, f, etc (we will demonstrate this in Section 5.2). Hence, if A is excessively small, a dual index may significantly outperform a primal index in query processing, after the latter has degraded seriously as time passes.

However, *a dual index never achieves the optimal efficiency at any time $t > 0$* (in other words, *when A is sufficiently high, at time t, a dual index always has worse query performance than a primal access method*). Specifically, Formula 20 is constantly larger than Formula 8, for all $t_c \in [T, 2T)$. This can be understood as follows. Remember that the optimal quality is reached, if and only if (i) all the objects are managed by a single structure, and (ii) the reference time of the structure equals the current time t_c. During $[T, 2T)$, condition (i) holds at time T or in the time interval $[T + 1/A, 2T)$, when $Index_1$ or $Index_2$ is the sole structure indexing the entire dataset, respectively. However, condition (ii) is always violated when (i) is fulfilled: at time T, $Index_1$ has reference time 0, whereas, during $[T+1/A, 2T)$, $Index_2$ has reference time T.

6 Discussion about the Update Performance

Some empirical evidence [11, 16] suggests that, as far as the existing solutions are concerned, dual indexes incur lower update cost than primal structures. This observation has, to some extent, misled the spatiotemporal community into the misconception that the primal technique is a worse approach in practice, despite its superiority in query performance. This misconception comes from the argument that, in a spatiotemporal application, there are much more updates than queries[5] such that the overall workload of a database is

[5]This is true. For example, in a highway monitoring system, a vehicle generates an update whenever its moving speed or direction has changed significantly. The update frequency of an entire dataset may be at the order of once per second, and is much

dominated by the overhead of updates, which advocates the employment of dual access methods.

In this section, we aim to eliminate the above misconception, and thereby, re-establish the practical importance of the primal methodology. Our discussion proceeds in three steps.

Commercial Importance of Optimizing Query Performance. Update overhead and query overhead are two types of cost with different influences on the commercial success of an application. Specifically, update performance limits the scalability of the database, in terms of the maximum number of objects that can be supported, subject to the system's computing capacities. Query performance, however, affects the service quality that can be offered to a customer, namely, how fast her/his look-up request can be processed.

Query cost actually has a more immediate impact on the public image of a system, since it can be directly noticed by customers when they are waiting for their results. Update cost, on the other hand, is "internal", because it is hidden from the public, i.e., an object does not need to know how fast its updates were handled. Therefore, minimization of query cost is crucial, even though it is not the bottleneck of the system's overall performance. In particular, even if an index structure is query-efficient but relatively update-expensive, it still has practical merits, as it helps to improve the "tangible" service quality of the system. Note that, the above discussion is valid under the circumstances that the system is able to handle all updates (i.e., no load shedding is necessary). This is highly possible, given the gigantic computing power of modern machines.

The Update Cost of a Perfect Primal Index. It is not correct to claim that the primal technique has worse update performance than the dual methodology, simply because the *known* primal structures are slower in updates than their dual counterparts. In other words, it could be just that a "perfect" primal index had not been discovered yet, and such a solution would have excellent update efficiency. In fact, the only rigorous way to conclude that the primal technique is inherently update-expensive, would be to establish a lower bound for its query cost, when the update overhead must be controlled under a certain limit [8]. For example, the conclusion would be convincing, if one could show that, when the update cost must be logarithmic to the dataset cardinality, the query overhead would have to exceed a large value. Interestingly, as explained next, our findings in the previous sections point to the opposite: a good primal index should have nice update efficiency, if it undergoes no structural deterioration.

An update involves an insertion and a deletion. Let us focus on deletions, because they are more expensive (an insertion can be completed by accessing, on average, a single path of the root to a leaf, as is a property of

higher than the query frequency.

the TPR-tree [17]). A deletion, specifically, is divided into two steps. The first one locates the leaf node that contains the object being deleted, whereas the second step removes the object from the node, and propagates the changes to the upper levels of the tree. We concentrate on the first step, which dominates the overall deletion overhead. (In particular, the second step often entails no I/O cost, since it usually affects only the path from the root to the leaf node accommodating the object, and the path has been accessed in the first step, and thus, still resides in the buffer. The second step incurs higher cost, only when a node underflow happens. Underflows, however, are provably infrequent, under the realistic condition that a deletion may occur in any leaf node with an equal probability. Specifically, as mentioned in Section 5.1.1, at each timestamp in expectation $A \cdot f$ objects are inserted and removed from a node, respectively.)

Let o be the object being removed. Locating the leaf node containing o requires, in the worst case, visiting all the leaf nodes whose MBRs, in the dual space at the insertion time t of o (t is smaller than the current time t_c), cover the dual representation of o at time t. This process could be expensive because, due to structural deterioration, the MBRs of many nodes may overlap, and o may happen to fall in their overlapping region. However, if the tree undergoes no deterioration, all the leaf nodes have disjoint MBRs in the dual space at time $t_c - 1$, and hence, their MBRs must also be disjoint in the dual space at time t (notice that the MBR of a node at an older timestamp must be contained by its MBR at a later timestamp). As a result, o is covered by the MBR of exactly one leaf node in the dual space at time t.

The above analysis indicates that, if no deterioration occurs in a primal index, only one leaf node needs to be accessed to discover the object in the tree. The analysis can be extended to the intermediate levels in a straightforward manner. It follows that, without structural deterioration, a single path from the root to the leaf level needs to be visited in a deletion. In this case, the deletion cost is no worse than that of any dual index in the literature. It is worth mentioning that, the discussion earlier assumes that conventional update algorithms, such as those developed in [17, 18], are applied. In fact, recognizing that indexes adapted from R-trees may incur large deletion overhead after structural degradation, the spatial community has developed several novel approaches (e.g., the bottom-up method [14] and the memo-method [21]) for alleviating this problem. Those approaches allow retrieval of the node containing an object to be deleted in $O(1)$ I/Os, and thus, further improve the update efficiency of a primal index.

The Periodic-Update Requirement of the Dual Technique. A dual method can be applied, only if all the objects obey a requirement: they must issue at least one update every T timestamps (we mentioned this at

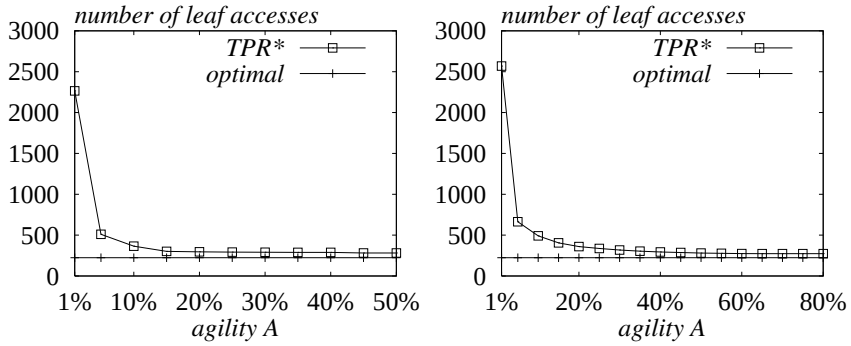
the end of Section 2.2), where T is the length of an update period. In practice, however, this requirement may not be fulfilled. For example, the next timestamp for an aircraft to issue an update, depends on its flight trajectory (e.g., no update is required, as long as the aircraft moves linearly), and may be larger than its previous update time by more than T timestamps. The primal technique does not suffer from this defect. Specifically, an object sends an update, only when it is absolutely necessary to do so — a velocity change has happened.

To remedy the defect, the dual technique takes a "mandatory approach", i.e., forcing an object to generate an update anyway at the end of a period, if it has not already done so earlier in this period. Although this approach guarantees the correctness of query results, it introduces another two drawbacks. First, it necessitates redundant updates, and hence, consumes a larger amount of network bandwidth. Second, it may render a dual index actually more update-expensive than a primal structure. This happens when the dataset agility A exceeds the deterioration threshold of the primal technique, and yet, is lower than $1/T$. As mentioned earlier, when no structural deterioration happens, the cost of a good primal index in handling a single update should be comparable to that of a dual structure. However, the dual structure must process a larger number of updates, since many objects, which do not generate voluntary updates, are forced to issue redundant updates. As a result, overall the dual technique entails higher maintenance overhead.

7 Numerical and Empirical Results

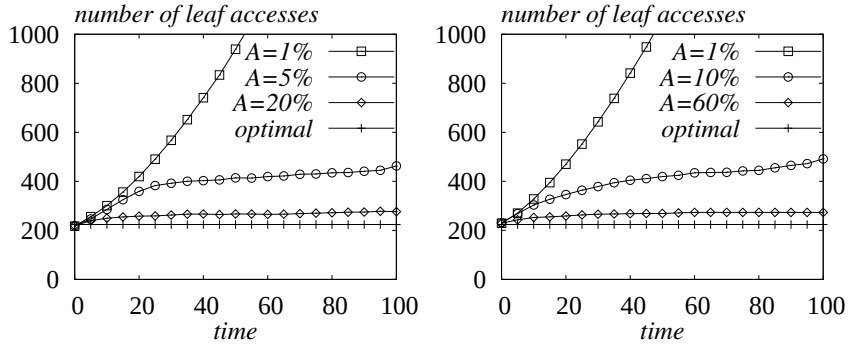
This section aims at achieving four purposes:

- Since some equations in our analysis cannot be transformed into closed forms, we solve them numerically, and reveal additional characteristics of the primal/dual techniques.
- We show that the "theoretical characteristics" can indeed be observed from the existing access methods.
- We evaluate the potential improvement for these indexes, by comparing their cost to the optimal performance.
- We apply our analytical findings to explain the experiment results in the previous work [11, 16, 18, 22], which seem to contradict each other.



(a) ps = 1024 (b) ps = 4096

Figure 5: TPR* quality at timestamp 100 ($H = 30$)



(a) ps = 1024 (b) ps = 4096

Figure 6: Typical TPR* deterioration ($H = 30$)

7.1 Settings

Our experiments examine the state-of-the-art primal and dual indexes: the TPR*-tree [18] and B^{dual}-tree [22], respectively. Since the B^{dual}-tree has not been compared with the (dual) solution of [13] previously, we also implement (an enhanced version of) that solution, referred to as R^{dual} in the sequel. The implementation is essentially an R-tree [2], whose update algorithms are adapted to minimize the metric of Equation 1. Each insertion/deletion is performed individually, i.e., no bulkloading is performed.

We generate 2D data conforming to the problem formulation in Section 3. The page size ps equals 1024 or 4096 bytes; accordingly, the average node fanout f (of all structures) equals 39 or 157, respectively. V is fixed to 0.01 (i.e., the largest object velocity is 100 times smaller than the length of a spatial dimension). For dual methods, T equals 20 (every object issues at least an update every 20 timestamps). The values of n (number of leaf nodes), A (agility) and H (horizon) will be clarified in individual experiments. Note that, once n is finalized, the value of N is also decided as $f \cdot n$. Each dataset evolves for a *history* of 100

timestamps ($A \cdot N$ objects are updated at each timestamp). Thus, the largest dataset occupies over 80 Giga bytes of space.

A *query workload* has 10000 point queries, whose locations are random in the spatial domain, and their timestamps are uniformly distributed in $[t, t + H]$, where t is the time of workload execution. The *quality* of an index is measured as the average number of leaf nodes accessed in answering a workload query.

7.2 Primal Deterioration

As analyzed in Section 5.1, the query cost of a primal index may continuously increase with time, if A is excessively low. However, the index retains the same efficiency at all times, after A reaches a certain threshold. The first set of experiments demonstrates the phenomenon on TPR*-trees. Towards this, we set n, H to 30k and 30 respectively, but vary A in a wide range. It is worth mentioning that, some combinations of A and H may not necessarily be realistic in practice. For example, $A = 50\%$ implies that each object is expected to issue an update every two timestamps, in which case it may not be useful to optimize a TPR*-tree for $H = 30$ future timestamps. Nevertheless, since our objective is to evaluate the derived theoretical findings, we examine those combinations anyway, in order to make sure that our experiments consider a wide spectrum of values for each parameter.

Figures 5a and 5b plot the TPR* quality at the end of history, as a function of A, when ps (page size) equals 1024 and 4096, respectively. The curve labeled as *optimal* represents the theoretical lower bound, computed with Equation 8. Clearly, the quality at time 100 is significantly worse than the optimal value for A = 1%, but gradually approaches the lower bound as A increases. The lower bound is identical in Figures 5a and 5b, because it relies on only V, H, and n (which are the same in the two figures).

In Figure 6a, we inspect the quality changes (in the experiment of Figure 5a) during timestamps 0-100, focusing on A = 1%, 5%, and 20%, respectively. The curves at these agilities illustrate 3 types of TPR*-degradation (the curve "optimal" has the same meaning as in Figure 5).

First, for exceedingly small A (e.g., 1%), the query overhead rapidly and continuously escalates with time, until object clustering in the TPR*-tree is completely random. Second, if A is low, but not exceedingly (e.g., 5%), the efficiency also degrades severely at the beginning. However, unlike the case A = 1%, the deterioration rate decreases with time, such that the query cost eventually stays constant at a high value. Third, as A grows further, the initial degradation becomes less obvious; furthermore, the overhead also

stabilizes faster. After A is sufficiently large (e.g., 20%), the query cost is always close to the lower bound.

Figure 6b presents similar results for A = 1%, 10%, 60% in the experiments of Figure 5b, confirming the same observations. Notice that, regardless of A and ps, the performance of TPR*-trees is nearly optimal at time 0.

7.3 Primal Degradation Agility

As elaborated in Section 5.1.2, the deterioration agility threshold, denoted as A_{thrs}, can be computed as follows. We first solve A from Equation 10, and then determine A_{thrs} as $\max\{A, 1/f\}$, where f is the average node fanout. In Table 2a, for ps = 1k, we list the theoretical thresholds for all the combinations of H and n, when they equal to 5 different values in $[10, 50]$ and $[10k, 50k]$, respectively. Table 2b demonstrates the results for ps = 4k.

The first important observation is that A_{thrs} is independent with the number n of leaf nodes (hence, also with the dataset cardinality). Instead, it only relies on H, or in other words, the "aspect ratio" between s and w, i.e., the extent lengths of a node on the spatial and velocity dimensions, respectively (as in Equation 5, $s/w = H/\sqrt{3}$).

The second crucial phenomenon is that, for $H \in [10, 30]$, A_{thrs} is significantly higher, when a larger page size is used. This fact negatively impacts the applicability of the primal technique in practice. Specifically, since ps = 4096 is "standard" in many database systems, a primal index should be used, only if a significant portion of the dataset generates updates at every timestamp. Otherwise, the index performance risks (gradual) deterioration, and eventually, ends up with unacceptable performance.

Finally, notice that A_{thrs} monotonically decreases as H increases. Remember that a large H optimizes queries in the "distant" future, and leads to nodes with long spatial extents (and short velocity extents). As a result, fewer objects can move out of the current SBR (of the node containing them) at the next timestamp, making it easier for the primal structure to retain its efficiency (see the analysis in Section 5.1).

Tables 2c and 2d present the actual deterioration agilities[6] of TPR*-trees, with respect to the same parameter combinations in Tables 2a and 2b. These actual thresholds are consistent with the facts revealed earlier from the theoretical ones. Note that, it is reasonable for an actual threshold to be (even much) higher than the

[6]We declare a TPR*-tree "deteriorated" if its quality at timestamp 100 exceeds the quality at time 0 by 25%. Given a pair of H and n, the threshold agility is the lowest agility at which TPR* deterioration is observed.

$n(N)\backslash H$	10	20	30	40	50
10k (390k)	11.8%	2.6%	2.6%	2.6%	2.6%
20k (780k)	11.8%	2.6%	2.6%	2.6%	2.6%
30k (1.17m)	11.8%	2.6%	2.6%	2.6%	2.6%
40k (1.56m)	11.8%	2.6%	2.6%	2.6%	2.6%
50k (1.95m)	11.8%	2.6%	2.6%	2.6%	2.6%

(a) ps = 1024 (theoretical)

$n(N)\backslash H$	10	20	30	40	50
10k (1.57m)	78%	56%	28%	4%	1%
20k (3.14m)	78%	56%	28%	4%	1%
30k (4.71m)	78%	56%	28%	4%	1%
40k (6.28m)	78%	56%	28%	4%	1%
50k (7.85m)	78%	56%	28%	4%	1%

(b) ps = 4096 (theoretical)

$n(N)\backslash H$	10	20	30	40	50
10k (390k)	40%	28%	20%	17%	15%
20k (780k)	40%	28%	20%	17%	15%
30k (1.17m)	40%	28%	20%	17%	15%
40k (1.56m)	40%	28%	20%	17%	15%
50k (1.95m)	40%	28%	20%	17%	15%

(c) ps = 1024 (TPR*)

$n(N)\backslash H$	10	20	30	40	50
10k (1.57m)	80%	68%	60%	54%	50%
20k (3.14m)	80%	68%	60%	54%	50%
30k (4.71m)	80%	68%	60%	54%	50%
40k (6.28m)	80%	68%	60%	54%	50%
50k (7.85m)	80%	68%	60%	54%	50%

(d) ps = 4096 (TPR*)

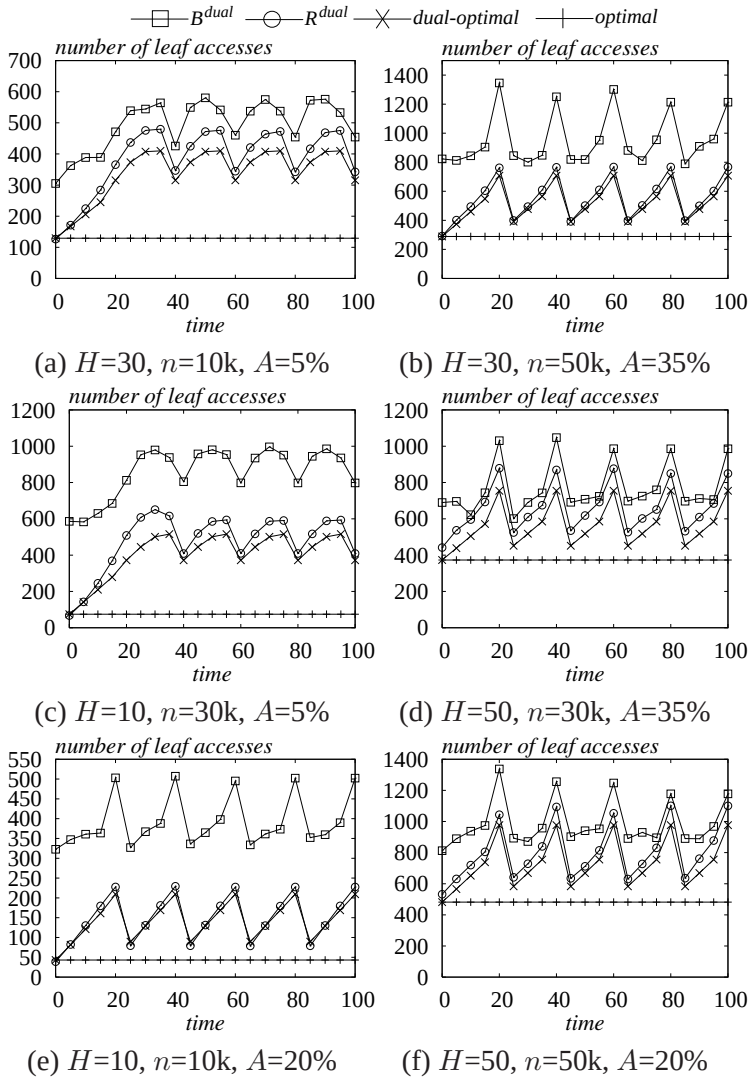
Table 2: Primal degradation agilities

corresponding theoretical A_{thrs}, since a TPR*-tree is not a "perfect" primal index.

7.4 Dual Periodic Behavior

We proceed to study the dual technique. For this purpose, ps is irrelevant; as shown in Section 5.2, the query cost of dual indexes is independent with the node fanout f, as long as the number n of leaf nodes is the same. The results presented in this subsection are obtained with $ps = 4096$.

To examine the influence of H, n, A, we choose 3 values for each parameter: H = 10, 30, or 50; n = 10k, 30k, or 50k; A = 5%, 20%, or 35%. Setting H to the median 30, Figure 7a (or 7b) plots the cost of B^{dual}- and



(a) H=30, n=10k, A=5% (b) H=30, n=50k, A=35%

(c) H=10, n=30k, A=5% (d) H=50, n=30k, A=35%

(e) H=10, n=10k, A=20% (f) H=50, n=50k, A=20%

Figure 7: Dual quality changes (any page size). Note that $\boldsymbol{n}$ is the number of *leaf nodes*; see Table 2 for the corresponding dataset cardinalities $\boldsymbol{N}$.

R^{dual}-trees as time evolves, using the smallest (or largest) values for the other two parameters. In Figures 7c and 7d, we fix n to the median 30k, and repeat the previous experiments, treating the other parameters in the same manner. Finally, Figures 7c and 7d demonstrate similar results with the median A = 20%.

Each of Figures 7a-7f also includes two reference curves *optimal* and *dual-optimal*, indicating different lower bounds. Specifically, *optimal* is the value of Equation 8, representing the general smallest overhead of both primal and dual indexes. On the other hand, *dual-optimal* corresponds to the value of Equation 20, i.e., the best efficiency of dual methods.

At timestamp 0, *optimal-dual* and *optimal* share the same value because, as explained in Section 4, primal and dual indexes have essentially equivalent forms, when they are just constructed. However, at any other time, *optimal-dual* is consistently worse than *optimal*, confirming the prediction in Section 5.3 that a dual index can never possess the optimal performance except at the initial timestamp.

Optimal-dual is periodic, but its shape varies depending on the concrete parameters. In each period, *optimal-dual* has a local minimum and maximum. In particular, the minimum is always achieved at the moment when the passive structure (recall that a dual index involves two structures, only one of which is active at any particular timestamp) becomes empty, i.e., the $1/A$-th timestamp of each period. On the other hand, the maximum occurs when the two structures have roughly an equal number of objects.

It is clear that the behavior of both B^{dual}- and R^{dual}-trees follows very closely *optimal-dual*, validating the correctness of our derivation. In particular, the R^{dual} cost approximates *optimal-dual*, suggesting that this index almost achieves the lower bound of dual access methods.

7.5 Resolving the Contradiction in the Experiments of Previous Work

There seems to be some inconsistency among the experiment results in the previous work [11, 16, 18, 22], which develops the most efficient spatiotemporal indexes. Specifically, the TPR*-tree in [18] has low and stable query cost, even after a long history. However, this is challenged in [11, 16], where TPR*-trees are significantly outperformed by dual-solutions STRIPES and B^x-trees, respectively. Recently, the relative superiority is again reversed in [22], where TPR*-trees are faster in query processing than all the dual indexes.

Equipped with the theoretical findings in this paper, we can easily explain the reasons underlying the above "inconsistency" (apart from the discrepancies of the implementations by different authors). That is, all the experiments advocating (or against) TPR* deploy disk pages of 1024 (or 4096) bytes, and datasets with large (or small) agilities. The agility exceeds (or falls far below) the degradation threshold in Table 2; hence, the TPR* exhibits excellent (or severely-deteriorating) performance.

In [18], the update interval of an object essentially follows a highly skewed distribution. Specifically, a majority of the objects issue velocity changes frequently, whereas a small subset of the database is not updated for a (very) long period. In [22], the objects have low velocities, which follow a Zipf distribution in $[0, 0.005]$ (skewed towards 0). Furthermore, each object is required to report its location every 25 timestamps. The effect is equivalent to soliciting an update from an object (frequently) whenever it moves a short distance.

Therefore, in [18, 22], the TPR*-cost grows in a way resembling curve A = 20% in Figure 6a.

On the other hand, the velocities of the objects in [11, 16] are around 3 times lower[7] than those used in Sections 7.2-7.4. An object issues an update (on average) every 60 timestamps, i.e., the dataset agility is $\frac{1}{60}$. This is analogous to an agility of $\frac{1}{60} \times 3 = 5\%$ in our settings, which is more than 10 times lower than the degradation agility threshold of TPR*-trees (recall that, the threshold is much higher for ps = 4096 than 1024). As a result, the behavior of TPR*-trees is similar to the A = 1% curve in Figure 6b.

8 Concluding Remarks

Both primal and dual methodologies have their important strengths and weaknesses in query processing. A primal index promises (much) lower query overhead (compare the curves *optimal* and *dual-optimal* in Figure 7), but may deteriorate continuously with time for some datasets, and become prohibitively expensive eventually. A dual access method, on the other hand, is "conservative"; its cost is far above the optimal value, but always remains reasonable, regardless of the data properties.

The dataset agility is the most influential factor on the relative superiority of the two methodologies. A primal index, unfortunately, is not suitable for large page sizes (e.g., beyond 4096), in which case the index is almost certain to keep deteriorating with time (unless the dataset agility is extremely high). In practice, if query efficiency is crucial (i.e., even the cost of *dual-optimal* is unacceptable, thus excluding dual methods), a possible solution is to re-construct a primal index periodically, after its cost has degraded to some extent.

Another pessimistic phenomenon revealed by our analysis is that, query processing is much harder in spatiotemporal databases than in the traditional spatial context. For example, when the underlying data is uniformly distributed, there may not be any $O(\log n)$ spatiotemporal queries at all. In other words, we should not expect any of the well-studied spatial operations (e.g., nearest neighbor search) to be solved by visiting just a few paths of a spatiotemporal index. All these operations require $\Omega(\sqrt{n})$ node accesses in expectation (see Section 4.2).

This work also initiates several directions for future research. First, as demonstrated in Table 2, the degradation agilities of the existing primal indices are higher than the theoretical thresholds. This phenomenon indicates that the primal technique still has much room for improvement. It would be worthy to investigate how to enhance the update algorithms of the TPR*-tree, in order to reduce its structural deterioration. Sec-

[7] In [11, 16], an object velocity takes one of 3 values 0.0007, 0.0015, and 0.003 with equal probability.

ond, our analytical findings suggest the possibility of combining the primal and dual techniques to design a good spatiotemporal index that takes the advantages of both techniques. Specifically, one may divide the underlying dataset into different clusters, based on the objects' update frequencies. To optimize query performance, a primal index may be constructed on the objects that issue updates frequently, whereas a dual structure may be created on the others. Finally, in this paper, we do not concern concurrent updates and queries. In practice, multiple queries and/or updates may be carried out on an index at the same time. Thus, it would not be sufficient to consider only the leaf level in performance analysis, since locking operations at the intermediate levels may have a major influence on the query/update efficiency as well. In that case, the relative superiority of the primal and dual techniques deservers further studies.

Acknowledgements

This work was partially supported by Grant CUHK 1202/06 from HKRGC.

References

[1] P. K. Agarwal, L. Arge, and J. Erickson. Indexing moving points. In *PODS*, pages 175–186, 2000.

[2] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger. The R*-tree: An efficient and robust access method for points and rectangles. In *SIGMOD*, pages 322–331, 1990.

[3] R. Benetis, C. S. Jensen, G. Karciauskas, and S. Salteni. Nearest and reverse nearest neighbor queries for moving objects. *To appear in VLDB Journal*.

[4] S. Berchtold, C. Bohm, D. A. Keim, and H.-P. Kriegel. A cost model for nearest neighbor search in high-dimensional data space. In *PODS*, pages 78–86, 1997.

[5] N. Bruno, S. Chaudhuri, and L. Gravano. Stholes: a multidimensional workload-aware histogram. In *SIGMOD*, pages 211–222, 2001.

[6] K. M. Elbassioni, A. Elmasry, and I. Kamel. An efficient indexing scheme for multi-dimensional moving objects. In *ICDT*, pages 425–439, 2003.

[7] J. Goldstein, R. Ramakrishnan, U. Shaft, and J.-B. Yu. Processing queries by linear constraints. In *PODS*, pages 257–267, 1997.

[8] J. M. Hellerstein, E. Koutsoupias, and C. H. Papadimitriou. On the analysis of indexing schemes. In *PODS*, pages 249–256, 1997.

[9] G. R. Hjaltason and H. Samet. Distance browsing in spatial databases. *TODS*, 24(2):265–318, 1999.

[10] H. Hu, J. Xu, and D. L. Lee. A generic framework for monitoring continuous spatial queries over moving objects. In *SIGMOD*, pages 479–490, 2005.

[11] C. S. Jensen, D. Lin, and B. C. Ooi. Query and update efficient B+-tree based indexing of moving objects. In *VLDB*, pages 768–779, 2004.

[12] G. Kollios, D. Gunopulos, and V. J. Tsotras. On indexing mobile objects. In *PODS*, pages 261–272, 1999.

[13] G. Kollios, D. Papadopoulos, D. Gunopulos, and J. Tsotras. Indexing mobile objects using dual transformations. *The VLDB Journal*, 14(2):238–256, 2005.

[14] M.-L. Lee, W. Hsu, C. S. Jensen, B. Cui, and K. L. Teo. Supporting frequent updates in r-trees: A bottom-up approach. In *VLDB*, pages 608–619, 2003.

[15] B.-U. Pagel, H.-W. Six, H. Toben, and P. Widmayer. Towards an analysis of range query performance in spatial data structures. In *PODS*, pages 214–221, 1993.

[16] J. M. Patel, Y. Chen, and V. P. Chakka. Stripes: An efficient index for predicted trajectories. In *SIGMOD*, pages 637–646, 2004.

[17] S. Saltenis, C. S. Jensen, S. T. Leutenegger, and M. A. Lopez. Indexing the positions of continuously moving objects. In *SIGMOD*, pages 331–342, 2000.

[18] 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.

[19] Y. Tao, J. Sun, and D. Papadias. Analysis of predictive spatio-temporal queries. *TODS*, 28(4):295–336, 2003.

[20] Y. Theodoridis and T. Sellis. A model for the prediction of R-tree performance. In *PODS*, pages 161–171, 1996.

[21] X. Xiong and W. G. Aref. R-trees with update memos. In *ICDE*, 2006.

[22] M. Yiu, Y. Tao, and N. Mamoulis. The B^{dual}-tree: Indexing moving objects by space-filling curves in the dual space. *To appear in VLDB Journal*.