Performance Analysis of R*-Trees with Arbitrary Node Extents

Yufei Tao and Dimitris Papadias

Abstract—Existing analysis for R-trees is inadequate for several traditional and emerging applications including, for example, temporal, spatio-temporal, and multimedia databases because it is based on the assumption that the extents of a node are identical on all dimensions, which is not satisfied in these domains. In this paper, we propose analytical models that can accurately predict R*-tree performance without this assumption. Our derivation is based on the novel concept of extent regression function, which computes the node extents as a function of the number of node splits. Detailed experimental evaluation reveals that the proposed models are accurate, even in cases where previous methods fail completely.

Index Terms—Database, spatial database, R-tree, cost model.

1 Introduction

The R-tree [14] is one of the most popular multidimensional access methods, currently incorporated in various commercial products such as Oracle and Informix. Object minimum bounding rectangles (MBRs) are grouped together in leaf nodes according to their spatial proximity, which are then recursively grouped in higher levels until the root contains a single node. Fig. 1 shows a simple 2D example where five objects (a, b, c, d, e) are clustered into two leaf nodes N_1 and N_2 that constitute the two entries in the root R. R-trees (like most spatial access methods) were motivated by the need to efficiently process window queries. The R-tree answers the query q in Fig. 1 as follows: The root is first retrieved and the entries (e.g., N_2) that intersect the range are recursively searched because they may contain qualifying objects (object e). Nonintersecting entries (e.g., N_1) are skipped.

In addition to "traditional" spatial applications, R-trees have also been widely used to index data from other domains. As an example, consider a banking system that records the historical changes of account balances as a result of withdrawals and deposits. Old versions of the records are not removed since possible queries may inquire about any time in history (e.g., find accounts with balances greater than 5k dollars during May or June). Each record is modeled as an interval, e.g., in Fig. 2a segments b_0, b_1, b_2, b_3 indicate that three deposits have been made to account b (current balance 5k dollars). Similarly, account d incurs one withdrawal, while a and c do not change during the recorded period. Fig. 2a also shows the MBRs of the leaf nodes N_1, N_2, N_3 in the corresponding R-tree. This approach (i.e., viewing each record as a 2D interval) has been adopted in numerous R-tree-based indexes [36], [21], [22], [7], [40] for temporal databases.

In spatio-temporal and multimedia databases (that manage large volume of moving objects), the most basic form of R-trees is the 3D R-tree, which considers time as just another dimension and integrates it in the tree construction along with the other dimensions. The movements of 2D objects are modeled as distinct boxes in the three-dimensional space. The temporal projection denotes the period when the corresponding object remains static, while the spatial projection of the box corresponds to the object's position and extent during this period. Whenever an object moves to another position, a separate box is created to represent its new static period, position, and extents. Fig. 2b extends the example of Fig. 1 assuming that at time t_1, objects d, and e issue location updates, which result in new versions d' and e', leading to another leaf node N_2'. Variations of the above idea have been exploited for indexing multimedia objects [48], trajectory [32], and historical information retrieval [38] for real-world data. Furthermore, the 3D R-tree has been used as a part of a multitree structure for aggregate processing in the context of spatio-temporal data warehouses [35].

A number of analytical models (reviewed in the next section) have been proposed to capture the performance of R-trees. *One basic assumption in these models is that the extents of a node are similar on all dimensions which, however, does not hold for temporal, spatio-temporal, and multimedia objects*. As shown in Fig. 2, for example, object "lifespans" (i.e., the projection of intervals/3D boxes on the time dimension) depend on the lengths of the periods that the corresponding objects retain their attributes (i.e., salary and positions in Figs. 2a and 2b, respectively). In particular, if a record does not change for a long time, then its lifespan will force its parent node to have (much) longer extent (on the temporal dimension) than those nodes containing only objects with short lifespans. *In such situations, the existing R-tree analysis is simply inapplicable*. Although the "elongated-node-extent" problem is well-known [22], [7], [40], predicting the R-tree performance in this scenario remains an open problem,

- *Y. Tao is with the Department of Computer Science, City University of Hong Kong, Tat Chee Avenue, Hong Kong. E-mail: taoyf@cs.cityu.edu.hk.*
- *D. Papadias is with the Department of Computer Science, Hong Kong University of Science and Technology, Clear Water Bay, Hong Kong. E-mail: dimitris@cs.ust.hk.*

Manuscript received 9 Oct. 2003; revised 5 Feb. 2004; accepted 10 Feb. 2004. For information on obtaining reprints of this article, please send e-mail to: tkde@computer.org, and reference IEEECS Log Number TKDE-0197-1003.

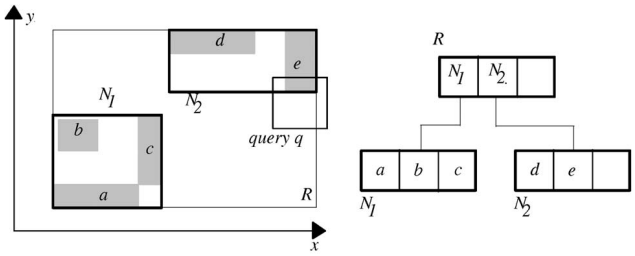

Fig. 1. R-tree for spatial data.

which seriously limits query optimization in the related systems.

This paper settles the problem for the R*-tree [10], the most efficient and widely-used variant of R-trees. In particular, we propose a cost model, based on the novel concept of *extent regression functions* (*ERFs*), that accurately captures the R*-tree performance for arbitrary data extents, independently of the concrete application domain. *ERFs* provide considerable insight into the behavior of R*-trees and motivate new optimization techniques, even in well-studied domains (e.g., spatial databases). Further, our analytical framework can be also applied to other data partition methods (e.g., the BKD-[28], X-[8], A-[41] trees, etc.).

The rest of the paper is organized as follows: Section 2 surveys existing approaches for R-tree analysis and elaborates why they fail to produce correct estimation for general data sets. Section 3 discusses our models, and Section 4 applies them to concrete applications. Section 5 presents an experimental study to confirm the effectiveness of the proposed models, while Section 6 concludes the paper with directions for future work.

2 Related Work

Since (as discussed in the next section) extent regression functions depend on the split algorithm of the corresponding multidimensional access method, in this section we first review the R*-tree split algorithm. Then, we survey the existing cost models for window queries and motivate our work by elaborating their deficiencies.

2.1 R*-Tree Split Algorithm

The R-tree clusters multidimensional objects by their spatial proximity. According to [19], [34], an efficient clustering should minimize metrics such as the overlap between MBRs, their perimeters, etc. Among the R-tree variants (e.g., [14], [39]), the R*-tree [10] is the most efficient (in terms query performance) mainly because of its improved split algorithm. Specifically, given a full node (i.e., the number of entries exceeds the node capacity by 1), the split algorithm distributes its entries into two new nodes aiming at minimizing the overlap between the resulting MBRs, while ensuring the minimum node usage of 40 percent.

To illustrate this, consider Fig. 3, where $r_1, r_2, \ldots, r_8$ correspond to the MBRs of entries in an overflowing node (i.e., the node capacity is 7). The algorithm first selects the split axis (among all dimensions) that leads to the smallest overall perimeter (see [10] for details). Let the chosen axis be the x-dimension in Fig. 3. Then, the entries are distributed based on the coordinates of their MBRs on this dimension. Specifically, all rectangles are sorted by the x-coordinates of their left boundaries (in the example, the sorted list is $\{r_1, r_2, \ldots, r_8\}$). Next, the algorithm attempts all possible divisions of the sorted list subject to the minimum node usage. In this case, a node should contain at least $3(\approx 40\% \cdot 7)$ entries; thus, possible distributions involve assigning the first 3, 4, or 5 entries (of

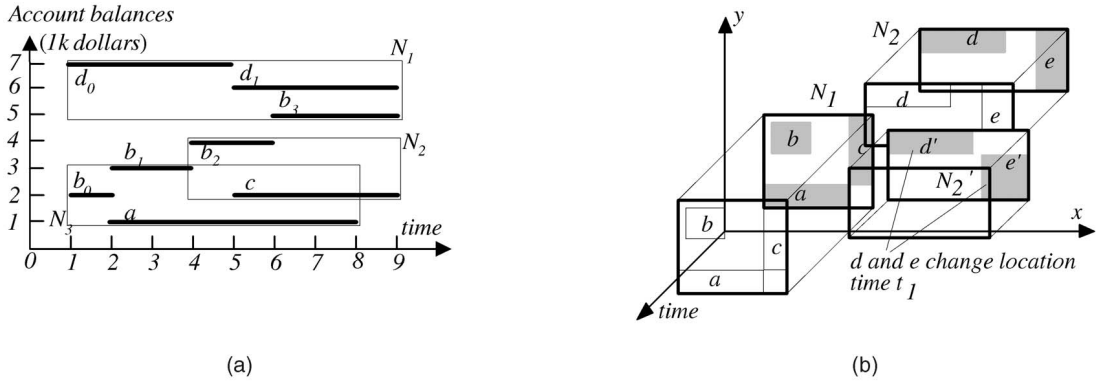

Fig. 2. Application of R-trees in temporal environments. (a) R-tree for temporal data and (b) R-tree for spatio-temporal data.

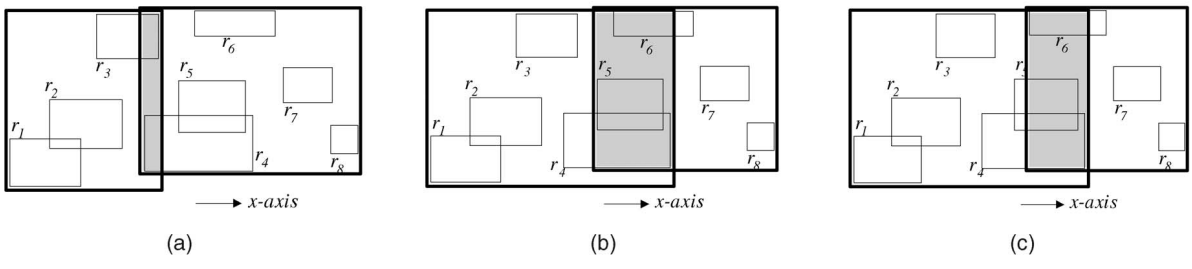

Fig. 3. The possible groupings. (a) 3-5 assignment, (b) 4-4 assignment, and (c) 5-3 assignment.

the list) into the first node and the remaining entries into the second one. Then, the final grouping is the one with minimum mutual overlap. Figs. 3a, 3b, and 3c illustrate the overlap regions (the dark areas) for these groupings. Clearly, the "3-5" assignment (Fig. 3a) achieves the smallest overlap.[1]

Notice that the extents of the new nodes are similar to those of the original node along the dimensions that do not incur split. For instance, in all cases of Fig. 3, the y-extents of the resulting nodes do not change significantly after the split. This is because the splitting is performed according to one axis only and, thus, has little discrimination effect along the other dimensions. This is a common property for many data-partitioning spatial indexes (e.g., the BKD-trees, X-trees, A-trees, etc.).

2.2 Existing Performance Studies

Faloutsos et al. [13] performed the first analysis for R-trees and R+-trees (by that time, R*-trees had not been proposed) focusing on 1D intervals. Later, [19] and [34] independently developed (2-1) that is the basis of most subsequent models. Specifically, for two d-dimensional rectangles r_1, r_2 randomly distributed in the unit data space $[0,1]^d$, the probability $P_{INTR}(r_1, r_2)$ that they intersect each other is:

$$P_{INTR}(r_1, r_2) = \prod_{i=1}^{d} (r_1.l_i + r_2.l_i), \tag{2-1}$$

where $r_1(_2).l_i$ denotes the MBR length along the ith $(1 \leq i \leq d)$ dimension for $r_1(_2)$. For a window query q, a node in the R-tree is visited if and only if its MBR intersects q. As a result, the probability that a node is accessed can be calculated with (2-1) provided that its extents are available. Based on this, the summation of the probabilities for all nodes in the tree gives the expected number of node accesses in answering q [19], [34], [18] as shown in following equation:

$$NA(q) = \sum_{i=1}^{M} P_{INTR}(r_i, q) = \sum_{i=1}^{M} \left[\prod_{j=1}^{d} (r_i.l_j + q.l_j) \right], \tag{2-2}$$

where M is the total number of nodes (of all levels), and r_i is the MBR of the ith node (NA stands for node accesses). The practical applicability of the above equation is limited because the extents of all nodes in the tree are usually not available in advance and, even if they are known (e.g., for static data), their summation may lead to expensive estimation overhead. To simplify this formula, Theodoridis and Sellis [46] assume a "regular uniform data model" where 1) all objects have the same extent length L_D along each dimension and 2) they are aligned into $N^{1/d}$ (N is the data set cardinality) rows/columns with a gap g_D between two consecutive rows/columns, as illustrated in Fig. 4. Given the density[2] D of the data set, the following formulae represent L_D and g_D:

1. The R* split algorithm performs another similar pass, which sorts the coordinates of the right boundaries. For the purpose of analysis, it suffices to consider only one pass due to symmetry.

2. The density D of a set of rectangles is defined as the average number of rectangles that contain a given point in the workspace. Equivalently, D can be expressed as the ratio of the sum of the areas of all rectangles over the area of the available workspace.

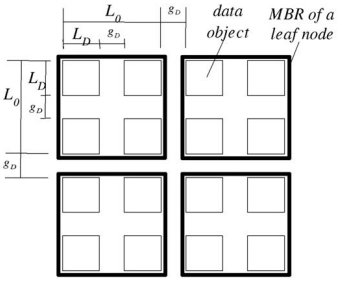

Fig. 4. The regular data model for 2D objects $(N = 16, f = 4)$.

$$L_D = \left(\frac{D}{N}\right)^{1/d}, \text{ and } g_D = \frac{1 - L_D \cdot N^{1/d}}{n^{1/d} - 1} = \frac{1 - D^{1/d}}{N^{1/d} - 1}. \tag{2-3}$$

Under this model, nodes at the same level have similar sizes (i.e., if nodes r_1, r_2 are at the same level, then $r_1.l_j = r_2.l_j$ for all $1 \leq j \leq d$) because the data clustering does not vary significantly across the data space. Let h be the height of the tree (i.e., leaves are at level 0), L_{ij} the average MBR extent of level-i nodes $(0 \leq i \leq h-1)$ along dimension $j (1 \leq j \leq d)$ and N_i the number of level-i nodes; then, (2-2) can be rewritten as:

$$NA(q) = \sum_{i=0}^{h-1} \left\{ N_i \cdot \left[\prod_{j=1}^{d} (L_{ij} + q.l_j) \right] \right\}. \tag{2-4}$$

Specifically, h and N_i are represented as

$$h = 2 + \left\lceil \log_f \frac{N/f}{b} \right\rceil, N_i = \frac{N}{f^{i+1}}, \tag{2-5}$$

where f is the average node fanout, i.e., the number of entries in a node (= 4 in Fig. 4), typically 69 percent [52] of the node capacity b. To facilitate the estimation of L_{ij}, Theodoridis and Sellis [46] further assume that objects in a leaf node are regularly aligned in $f^{1/d}$ rows and columns. As a result, each node MBR is a square defined by the extents of $f^{1/d}$ entries as well as the $(f^{1/d} - 1)$ gaps among them. Therefore, the extent L_0 of leaf nodes on each dimension is:

$$L_0 = (L_D + g_D) \cdot \left(f^{1/d} - 1\right) + L_D. \tag{2-6}$$

This process can be applied recursively to higher levels. Specifically, the MBRs of level-0 nodes can be regarded as a set of aligned rectangles with gaps g_D (i.e., same as the data objects) between rows or columns. Thus, following the same reasoning, the node extent L_{i+1} at tree level $i+1$ can be obtained from L_i as:

$$L_{i+1} = (L_i + g_D) \cdot \left(f^{1/d} - 1\right) + L_i. \tag{2-7}$$

Substituting these estimates for node extents in (2-4), we can predict the number of node accesses of a window query as a function of the cardinality N, dimensionality d, node capacity b, and data density D.

The above analysis can be extended to nonuniform data (where the regular modeling no longer holds) with histograms, based on the rationale that objects within a

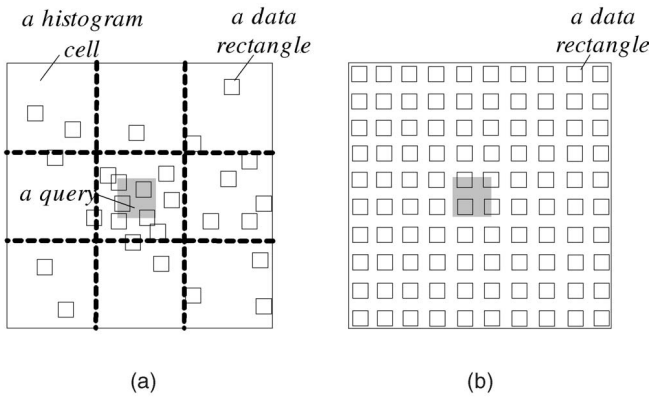

Fig. 5. Handling nonuniform distribution with histogram. (a) A query for nonuniform data ($H = 3$) and (b) the conceived uniform data (100 rectangles).

sufficiently small region are almost uniform, even though the overall distribution may deviate significantly. Theodoridis et al. [47] partition the data space into $H \cdot H$ regular cells (H is the resolution) and store the following statistics in each cell c: 1) the number N_c of objects whose MBR centroids fall in the cell and 2) the density D_c of these objects. Consider, for example, the query in Fig. 5a that falls in cell c_{22} (the subscript indicates the second row and column) with cardinality and density N_{22} and D_{22}, respectively. Its cost is approximated using a conceived uniform data set (Fig. 5b) with density D_{22} and cardinality $N_C \cdot H^2$. The reasoning is that the query only visits the nodes containing objects in c_{22}, and the extents of these nodes are similar to those on the conceived uniform data set. Note that the properties (i.e., cardinality and density) of the conceived data set depend on the query location. For example, for a query in c_{11} (i.e., the cell at the first row and column), the corresponding uniform data set is much sparser, and as a result, a different estimate is obtained. In general, if the query window intersects multiple cells, the average density of these cells is used instead. Jin et al. [17] present an interesting solution for packed R-trees [20] (i.e., assuming static data known in advance) using enhanced histograms. In this work, we focus on dynamic trees due to their higher importance in practice.

Faloutsos and Kamel [20] point out that the distributions of some real point data sets can be described using a simple equation called the *power law*. Based on this observation, they propose a method to capture R-tree performance using the *fractal dimension*, namely, the intrinsic dimension of the data set. Proietti and Faloutsos [30] extend the idea to rectangular objects obeying the *regal law*, i.e., the probability that the area of an object is greater than a certain value ξ equals $A^{-\xi}$, where A is a constant. While they claim that this law is satisfied in many real spatial data sets, there is no evidence that it holds for temporal and spatio-temporal data (actually the results of [45] suggest a different distribution in these scenarios).

In addition, there is a considerable amount of work on window query selectivity estimation (i.e., predicting the output size). Relevant methods adopt various multidimensional histograms [2], [1], [15], [6], *sampling* [4], [16], [49], *kernel estimation* [9], *singular value decomposition* [31], *compressed grid* [27], [23], [26], [51], *sketches* [42], *maximal independence* [11], *fractal* [5], [29], and *Euler formula* [17], [37], [25]. When the query selectivity (equivalently, the number n of objects satisfying the query) is available, a simple (but rather coarse) estimate for the query cost is $\sum_{i=0\sim(h-1)} \lceil n/f^{i+1} \rceil$, where f is the node fanout and h is the height of the tree [46], [2]. The rationale of this formula is that a node contributes f entries if its MBR is totally contained in the query region.

2.3 Shortcomings of Existing Analysis

Based on the above discussion, there exist three general methodologies for predicting the R-tree node accesses when answering window queries:

1. the Theodoridis and Sellis model for node extents combined with (2-2) for node access probability (TS for short),
2. techniques based on power laws (we call them *fractal*), and
3. techniques based on selectivity estimation and the formula $\sum_{i=0\sim(h-1)} \lceil n/f^{i+1} \rceil$ (we call them collectively as *selectivity* methods).

Each methodology has some serious shortcomings in practice.

First, the assumption of TS that the extents of a node are identical in all dimensions, as mentioned in the introduction, is not satisfied in a wide range of applications. In temporal databases, for instance, the temporal extents of R-tree nodes are determined by the lifespans of the objects that they contain and may vary significantly. Specifically, the extents should *follow certain probabilistic distribution that is related to the length distribution of objects' lifespans*. Consequently, the node sizes can no longer be captured by a simple formula such as (2-7).

Second, the applicability of *fractal* is limited to data sets that satisfy certain laws. In case of rectangle data sets, [30] assumes that 1) all the data objects have fixed aspect ratio (i.e., the ratio between its height and width) and 2) a node has similar extents on all axes (similar to [46]). Even if these conditions are satisfied (which is not the case for temporal or spatio-temporal data), fractal techniques produce a single estimate that corresponds to the average cost for all possible queries in the data space (regardless of the data distribution). Since queries at various positions lead to different costs, a single estimate for all queries may result in large individual errors.

Third, the correctness of *selectivity* relies on the assumption that the MBRs of most accessed nodes are totally contained in the query window, which is not true if 1) the query is not sufficiently large, or 2) node extents are elongated on particular axes. Finally, it is worth pointing out that none of the three methods distinguishes among the various R-tree variants (i.e., the same model is supposed to work for all different types of R-trees), although it is well-known that several variants have large performance differences. Motivated by these observations, in the next section, we present a cost model that focuses on R*-trees and provides accurate estimation, independently of the application domain.

TABLE 1
List of Symbols

Symbol	Description
h	Height of the R*-tree
N	Total number of objects in the dataset
N_i	Number of level i nodes (the leaf level is 0)
b	Node capacity
f	Average fanout
L_{ij}	Average extent on the jth axis of a level-i node
s_i	Total number of splits a level-i node incurs
s_{ij}	Number of splits on the jth axis a level-i node incurs
F_i	The length distribution function along the ith axis
ERF_i	The extent regression function along the ith axis

3 R*-Tree Analysis with *ERF*s

For our analysis, we assume that each data object is a d-dimensional rectangle r such that: 1) its center location follows certain distribution (called the *location distribution* in the sequel) in the unit data space $[0,1]^d$ and 2) the length $r.l_i$ of its extent along the ith dimension is decided by the probability distribution function F_i (referred to the *length distribution*), or specifically, $F_i(\xi)$ equals the probability $P\{r.l_j \leq \xi\}$ that $r.l_i$ is no larger than a certain length ξ. The objective is to predict the number of R*-tree node accesses in answering a window query q. Compared with existing analysis, the introduction of the length distribution increases the applicability of our method by allowing objects to have arbitrarily different sizes. We start with an overview of the proposed framework, focusing on uniform data, and then extend the solution to general location distributions with histograms. Table 1 contains the primary symbols to be used in our analysis (symbols that have not appeared yet, will be elaborated shortly).

3.1 Framework Overview

Similar to the TS model, our analysis is based on (2-4), and aims at estimating the average node extent L_{ij} at each level $i(0 \leq i \leq h-1)$ and dimension $j(1 \leq j \leq d)$. Observe that the MBR of a leaf node n_L can be regarded as the result of a series of splits from the root, whose MBR covers the entire data space (in particular, each split is performed along a single dimension). Let $n_L.s_{0i}$ (the subscript 0 means leaf level) be the number of times that a split (on a leaf node n_L) occurs along the ith dimension, and $\{n_L.s_{01}, n_L.s_{02}, \ldots, n_L.s_{0d}\}$ be the *split log*[3] of n_L. Fig. 6 shows an example of incrementally inserting 12 rectangles (block capacity $b = 3$). In Fig. 6a (after three objects have been inserted), the tree consists of a single root node R with split log $\{0,0\}$ (for x-, y-axes, respectively), meaning that no split has been performed on either dimension. In Fig. 6b, R splits (on the x-axis) into A and B whose split logs are then $\{1,0\}$. Note that the extents of both A and B are about half that of R on the x-axis and approximately the same on the y-axis. Fig. 6c shows the situation (after all insertions) where A and B have split along the y-axis, leading to four leaf nodes C, D, E, F with split logs $\{1,1\}$. Each MBR covers about $1/4$ of the data space.

3. The concept of split log can be regarded as a simplified "split history" as applied in X-trees [8].

Observation 1. *For uniform data, each leaf node splits about the same number of times along each dimension, or, formally, for any two leaf nodes n_{L1}, n_{L2} and any dimension $i(1 \leq i \leq d), n_{L1}.s_{0i} \approx n_{L2}.s_{0i}$.*

The reasoning behind the above observation is that, for uniform location distribution, data characteristics are the same throughout the data space and, thus, each leaf node should have similar split behavior. As a corollary, every leaf node incurs approximately the same number $s_0 (= s_{01} + s_{02} + \ldots + s_{0d})$ of splits (on all dimensions). Particularly, the relation between the number N_0 of leaf level nodes and s_0 is:

Observation 2. $s_0 \approx \log_2 N_0$.

To explain this, we borrow the concept of the *split tree* [8], i.e., a binary tree where leaf (*white*) nodes record the ids of leaf nodes of the R-tree. A *black* node, on the other hand, represents a historical leaf node that has disappeared after splitting into two nodes, whose ids are associated with the children of that black node. Fig. 7 shows the split tree for the R-tree in Fig. 6 (the letters in brackets correspond to the ids of the associated R-tree nodes). Black nodes a and b, for example, are children of node r, indicating that the root R splits into leaf nodes A and B (Fig. 6b). It is clear that the number of edges on the path from the root to a leaf node corresponds to the number of splits this leaf has ever incurred. According to Observation 1, this number should be approximately the same for all the leaf nodes or, equivalently, the total number s_0 of splits for a leaf node corresponds to the height (= 2 in this example) of the split tree. On the other hand, since N_0 equals the number of white nodes (each corresponding to a leaf node in the R-tree), the height of the split tree is represented as $\log_2 N_0$, indicating that $s_0 \approx \log_2 N_0$. The implication is that the total number of split times for a leaf node depends only on the data set cardinality N and the node capacity b (i.e., as shown in (2-5), $N_0 = N/f$ while $f = 0.69 \cdot b$).

Observation 3 (The min-marg rule). *The final clustering of an R-tree aims at minimizing the sum of perimeters of all leaf MBRs.*

The min-marg rule is consistent with the current understanding of the properties of a "good" R-tree. As revealed in [34], [20], [33], the clustering in an R-tree should minimize both the areas (or volumes for higher-dimensional spaces) and the perimeters of the leaf nodes MBRs. Note that

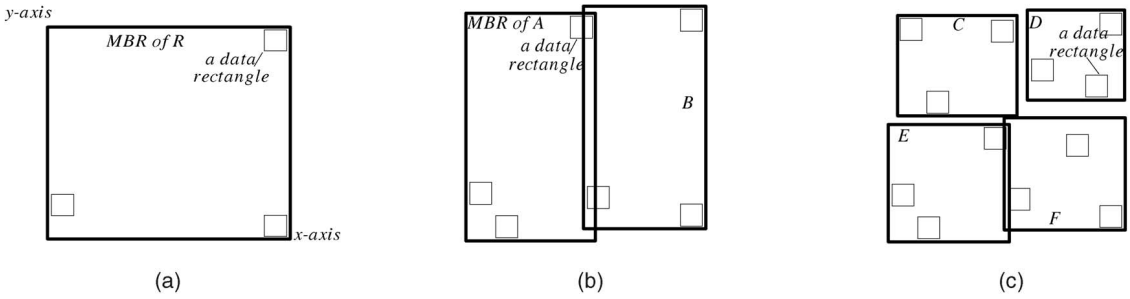

Fig. 6. The split axis. (a) Root is leaf, (b) two leaf nodes, and (c) four leaf nodes.

minimizing perimeters is more important because rectangles with smaller perimeters usually have smaller areas (but not the opposite).

Next, we introduce the *extent regression function*, which returns the extents of a node as a function of the number of splits. In particular, each dimension $i(1 \leq i \leq d)$ has an independent ERF_i which relies on the corresponding length distribution F_i. Notice that ERFs provide a natural way to describe MBR extents. Specifically, if we know that a leaf node has split s_{0i} times on the ith axis, then its extent L_{0i} on this dimension is given by $ERF_i(s_{0i})$. In general, given a split log $\{s_{01}, s_{02}, \ldots, s_{0d}\}$, the perimeter of a leaf node equals $\sum_{i=1\sim d} ERF_i(s_{0i})$. Recall that, the objective of our analysis is to obtain L_{0i} for all dimensions $(1 \leq i \leq d)$. Hence, equipped with $ERFs$, we reduce the problem into that of estimating the split log $\{s_{01}, s_{02}, \ldots, s_{0d}\}$, which can, in turn, be transformed into a constrained optimization problem (taking into account Observation 3).

Observation 4. *The split log $\{s_{01}, s_{02}, \ldots, s_{0d}\}$ of a leaf node minimizes the objective function $\sum_{i=1\sim d} ERF_i(s_{0i})$ under the constraint that $\sum_{i=1\sim d} s_{0i} = \log_2 N_0$, where N_0 is given in (2-5).*

We motivate the observation using Fig. 6 as an example. As will be shown later, for uniform rectangle data with small extents, $ERF(s_{0i}) = 1/2^{s_{0i}}$. On the other hand, possible permutations of (s_{01}, s_{02}) subject to the constraint $\sum_{i=1\sim d} s_{0i} = \log_2 4 = 2$ include $\{(0,2),(1,1),(2,0)\}$. It can be verified that (1, 1) minimizes the function $\sum_{i=1\sim d} ERF_i(s_0 i)$, which means that a leaf MBR should split once along each dimension (leading to extent $1/2$) as in Fig. 6c. Note that the above discussion also applies to nodes of higher levels $i(1 \leq i \leq h-1)$. Specifically, let s_{ij} be the total number of splits for a level-i node along the jth dimension; then, L_{ij} is estimated as $ERF_i(s_{ij})$. Further, since the min-marg rule also holds for higher levels, estimating the split log is equivalent to minimizing $\sum_{i=1\sim d} ERF_i(s_{ij})$ with the constraint $\sum_{j=1\sim d} s_{ij} = \log_2 N_i$, where N_i is the number of level-i nodes computed by (2-5). In the sequel, we first discuss the derivation of ERFs, and then elaborate on the concrete algorithms used for estimating the split log in Section 3.3.

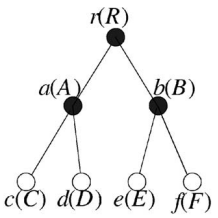

Fig. 7. Split tree for the R-tree of Fig. 6.

3.2 Derivation of *ERF*s

The derivation of $ERFs$ depends on the split algorithm of the concrete index structure. For R*-trees, the most important step in the derivation is to solve the following probability problem:

Problem 3.1. *Let*

$$[I_1.s, I_1.e], [I_2.s, I_2.e], \ldots, [I_{b+1}.s, I_{b+1}.e]$$

be $b+1$ independent 1D intervals such that b is the node capacity and $I_i.s(I_i.e)$ is the coordinate of the starting (ending) point of I. For each interval $I_i(1 \leq i \leq b+1)$:

1. *Its length $I_i.l = I_i.e - I_i.s$ satisfies the length distribution F (i.e., $F(\xi)$ equals the probability $P\{I_i.l \leq \xi\}$).*
2. *Its starting point $I_i.s$ distributes uniformly inside $[0, L - I_i.l]$, where L is a constant (that corresponds to the extent of a node to be split).*
3. *$I_i.s \leq I_{i+1}.s$ for two consecutive intervals I_i and I_{i+1} (i.e., all intervals are sorted by their starting points).*

Assume these intervals are split into two nodes, such that the first node contains $I_1, I_2, \ldots, I_m$, and the second one contains $I_{m+1}, \ldots, I_{b+1}$, for some constant $m \in [1, b+1]$. Let M_e be the largest ending point of the first m intervals: $M_e = \max_{i=1,\ldots,m}(I_i.e)$. The goal is to compute the expected values $\mathbf{E}(M_e)$ of M_e, and $\mathbf{E}(L - I_{m+1}.s)$ of $L - I_{m+1}.s$.

Next, we explain the relationship between Problem 3.1 and the ERF derivation. Consider, for instance, an overflowing node that splits along dimension i, on which its extent has length L. Intervals $I_1, I_2, \ldots, I_{b+1}$ correspond to the extents of the entries (on dimension i). Without loss of generality, assume that the left boundary of $I_1.s$ is at coordinate 0. The R* split algorithm (reviewed in Section 2.1) sorts the entries by $I_i.s$ in ascending order and groups the first m entries ($0.4b \leq m \leq 0.6b$ to ensure 40 percent node usage) into a new node and the remaining $(b+1-m)$ entries into another one. Thus, the extent of the first node is determined by the maximum of the ending points of $I_1, I_2, \ldots, I_m$, (i.e., M_e) and that of the second node by the starting point of I_{m+1}. This is illustrated in Fig. 8 $(b=4)$, where five entries are distributed into two nodes with $m=3$ and $b+1-m=2$

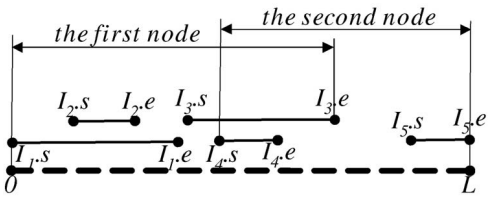

Fig. 8. The splitting of 1D intervals.

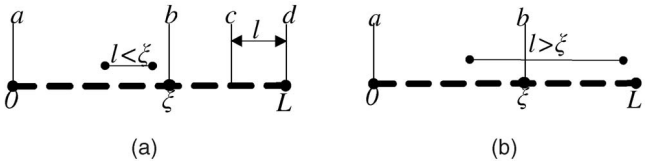

Fig. 9. Proof of Lemma 3.1. (a) $0 \le l \le \xi$ and (b) $\xi < l \le L$.

entries, respectively. The extent length of the first node equals M_e ($= I_3.e$, the maximum of $I_1.e$, $I_2.e$, $I_3.e$), and that of the second node equals $L - I_{m+1}.s$ ($= L - I_4.s$). Hence, the objective of Problem 3.1 is to derive the expected lengths of the resulting two nodes.

We start with the derivation of $\mathbf{E}(L - I_{m+1}.s)$ because it is simpler. Since $\mathbf{E}(L - I_{m+1}.s) = L - \mathbf{E}(I_{m+1}.s)$, it suffices to derive $\mathbf{E}(I_{m+1}.s)$. For this purpose, we first compute the probability $P\{I.s \le \xi\}$ that the starting point of any interval, which satisfies conditions 1 and 2) of Problem 3.1, is before coordinate ξ, subject to the length distribution $F(\xi)$.

Lemma 3.1. $\mathrm{P}\{I.s \le \xi\} = \int_0^\xi f(l)\frac{\xi}{L-l}dl + \int_\xi^L f(l)dl$, *where* $f(\xi)$ *is the probability density function of* $F(\xi)$.

Proof. Assuming, without loss of generality, that interval I has length $l (0 \le l \le L)$, we consider two cases: 1) $0 \le l \le \xi$, and 2) $\xi < l \le L$, and represent $\mathrm{P}\{I.s \le \xi\}$ as the sum of two conditional probabilities:

$$\mathrm{P}\{I.s \le \xi\} = \mathrm{P}\{I.s \le \xi | 0 \le l \le \xi\} + \mathrm{P}\{I.s \le \xi | \xi < l \le L\}. \tag{3-1}$$

Notice that since I is completely inside the range $[0, L]$, $I.s$ must fall in the range $[0, L - l]$ (i.e., between a and c in Fig. 9a).Since $I.s \le \xi$ if and only if $I.s$ lies in the range $[0, \xi]$, for case 1, we have:

$$\begin{aligned}\mathrm{P}\{I.s \le \xi \text{ and } 0 \le l \le \xi\} &= \int_0^\xi f(l)\mathrm{P}\{I.s \le \xi | I.l \le L\}dl \\ &= \int_0^\xi f(l)\frac{\xi}{L-l}dl.\end{aligned}$$

When $l > \xi$ (case 2)), on the other hand, the starting point $I.s$ is guaranteed to be less than ξ (Fig. 9b). So,

$$\mathrm{P}\{I.s \le \xi \text{ and } \xi < l \le L\} = \mathrm{P}\{\xi < l \le L\} = \int_\xi^L f(l)dl.$$

Thus, Lemma 3.1 holds. □

Using Lemma 3.1, we derive the probability $P\{I_{m+1}.s \le \xi\}$ that the starting points of at least $m+1$ (out of $b+1$) intervals are before ξ. Since the probability that exactly $i (m+1 \le i \le b+1)$ intervals fall in the range (while the other $b+1-i$ intervals do not) is

$$(P\{I.s \le \xi\})^i \cdot (1 - P\{I.s \le \xi\})^{b+1-i},$$

we have

$$\begin{aligned}&P\{I_{m+1}.s \le \xi\} \\ &= \sum_{i=m+1}^{b+1} \binom{b+1}{i} (P\{I.s \le \xi\})^i (1 - P\{I.s \le \xi\})^{b+1-i}.\end{aligned}$$

Therefore, $\mathbf{E}(I_{m+1}.s)$ can be represented as:

$$\mathrm{E}(I_{m+1}.s) = \int_0^L \xi \cdot \frac{d\mathrm{P}\{I_{m+1}.s \le \xi\}}{d\xi} d\xi. \tag{3-2}$$

Having solved $\mathbf{E}(L - I_{m+1}.s)$, we proceed to discuss the other half of Problem 3.1, i.e., deriving $\mathbf{E}(M_e)$. An accurate solution, however, is complex and requires considerable evaluation time (therefore, it is inappropriate for query optimization). Instead, we follow an alternative approach that provides approximate results with arbitrary precision, by studying the discrete version of Problem 3.1:

Problem 3.2 (discrete version of Problem 3.1). *Assume that the range* $[0, L]$ *is divided into* Λ *equal partitions with* $\Lambda + 1$ stamps *(i.e., the ith stamp is at coordinate* $i \cdot L/\Lambda$ *for* $0 \le i \le \Lambda$*). Let*

$$[I_1.s, I_1.e], [I_2.s, I_2.e], \ldots, [I_{b+1}.s, I_{b+1}.e]$$

be $b+1$ *intervals such that:*

1. *For each interval* $I_i (1 \le i \le b+1)$*, its end points* $I_i.s$ *and* $I_i.e$ *fall on these stamps.*
2. *Its length* $(I_i.e - I_i.s)$ *follows distribution* F.
3. *Its location distributes uniformly inside the range* $[0, L]$.
4. $I_i.s \le I_{i+1}.s$*, for two consecutive intervals* I_i *and* I_{i+1}.

The goal is to compute the expected value $\mathbf{E}(M_e)$ *for* M_e*, where* $M_e = \max_{i=1,\ldots,m}(I_i.e)$*, and* m *is a constant in* $[1, b+1]$.

It is easy to see that the value of $\mathbf{E}(M_e)$ obtained from the above problem equals that in Problem 3.1 when $\Lambda \to \infty$ (in fact, our experiments show that the approximated values are fairly close with $\Lambda = 100$). On the other hand, Problem 3.2 is easier because, intuitively, we only have a finite number of possible positions for the starting/ending points of each interval (while the number is infinite in Problem 3.1).

Similar to M_e, we define $M_s = \max_{i=1,\ldots,m}(I_i.s)$. To derive $\mathbf{E}(M_e)$, Lemma 3.2 computes, given two integers $0 \le \lambda \le \xi \le \Lambda$, the probability $P\{M_s = \lambda \cdot L/\Lambda, M_e \le \xi \cdot L/\Lambda\}$ that M_s lies exactly at the λth stamp and M_e lies at or before the ξth stamp.

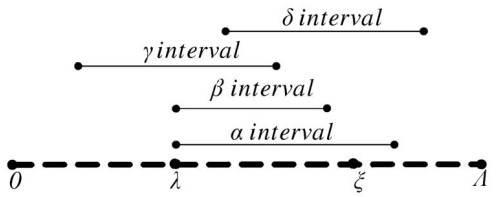

Fig. 10. Classifications of the intervals.

Lemma 3.2.

$$P\{M_s = \lambda L/\Lambda, M_e \le \xi L/\Lambda\} = \sum_{k=1}^{m-1} \sum_{j=m-k}^{b+1} \sum_{i=m-k-j}^{b+1} \left[\begin{matrix} C(b+1,i) \cdot C(b+1-i,j) \cdot C(b+1-i-j,k) \\ \cdot \alpha(\lambda,\xi)^i \cdot \beta(\lambda,\xi)^j \cdot \gamma(\lambda,\xi)^k \cdot \delta(\lambda,\xi)^{b+1-i-j-k} \end{matrix} \right],$$

where

$$C(u,v) = \begin{cases} \binom{u}{v} & \text{if } (u > v) \\ 0 & \text{otherwise,} \end{cases}$$

$$\alpha(\lambda,\xi) = \begin{cases} 0 & \text{if } \xi = \Lambda \\ \sum_{l=\xi+2-\lambda}^{\Lambda+1-\lambda} \frac{f_\Lambda(l)}{\Lambda+2-l} & \text{otherwise,} \end{cases}$$

$$\beta(\lambda,\xi) = \sum_{l=1}^{\xi+1-\lambda} \frac{f_\Lambda(l)}{\Lambda+2-l},$$

$$\gamma(\lambda,\xi) = \begin{cases} 0 & \text{if } \lambda = 0 \\ \sum_{l=1}^{\xi-\lambda+2} \frac{f_\Lambda(l)}{\Lambda+2-l} & \text{if } \lambda = 1 \\ \sum_{l=1}^{\xi-\lambda+2} f_\Lambda(l) \frac{\lambda}{\Lambda+2-l} + \sum_{l=\xi-\lambda+3}^{\xi+1} f_\Lambda(l) \frac{\xi+2-l}{\Lambda+2-l} & \text{otherwise,} \end{cases}$$

$$\delta(\lambda,\xi) = \begin{cases} 0 & \text{if } \lambda = \Lambda \\ \sum_{l=1}^{\Lambda-\lambda} f_\Lambda(l) \frac{\Lambda+1-\lambda-l}{\Lambda+2-l} & \text{otherwise,} \end{cases}$$

and $f_\Lambda(\varphi)) = F(\varphi \cdot L/\Lambda) - F[(\varphi-1) \cdot L/\Lambda]$.

Proof. We classify the intervals into four categories (Fig. 10), based on their relative positions with respect to the λth and ξth stamps (referred to as stamps λ and ξ in the sequel). The α group includes intervals that start at λ, and end after ξ. The α group involves intervals that also start at λ, but end at or before ξ. Intervals in the γ group start before λ, and end at or before ξ. The δ group contains intervals that start after λ. It is clear that these four groups are mutually disjoint, i.e., there cannot be any interval appearing in more than one group.

An α interval satisfies the conditions: 1) its starting point locates at ξ (among all possible positions in $[0, \Lambda]$), and 2) its length is within the range $[\lambda+1-\xi, \Lambda-\xi]$. Since the probability that the interval length equals l is $f_\Lambda(l)$, it is easy to verify that the probability that an interval becomes an α interval (subject to parameters ξ and λ) is given by $\alpha(\xi,\lambda)$ as represented in the lemma. Similarly, $\beta(\xi,\lambda)$, $\gamma(\xi,\lambda)$, and $\delta(\xi,\lambda)$ denote the probabilities that an interval belongs to β, γ, and δ categories.

Assume, without loss of generality, that there are i, j, k, and $b+1-(i+j+k)$ intervals in groups α, β, γ, δ, respectively, where $b+1$ is the total number of intervals. In the sequel, we refer to (i,j,k) as an *arrangement*. An arrangement is *legal* if the resulting intervals satisfy the condition $\{M_s = \lambda \cdot L/\Lambda, \text{ and } M_e \le \xi \cdot L/\Lambda\}$, where $M_s = \max_{p=1,\ldots,m}(I_p.s)$ and $M_e = \max_{p=1,\ldots,m}(I_p.e)$. For a legal arrangement,

1. $k < m$ (otherwise, $M_s < \xi \cdot L/\Lambda$),
2. $i+j+k \ge m$ (otherwise, $M_s > \lambda \cdot L/\Lambda$),
3. $j+k \ge m$ (otherwise, $M_e > \xi \cdot L/\Lambda$).

Note that the probability of an arbitrary arrangement is:

$$\binom{b+1}{i} \cdot \binom{b+1-i}{j} \cdot \binom{b+1-i-j}{k} \cdot \alpha(\xi,\lambda)^i \cdot \beta(\xi,\lambda)^j \cdot \gamma(\xi,\lambda)^k \cdot \delta(\xi,\lambda)^{b+1-i-j-k}.$$

Consequently, $P\{M_s = \lambda \cdot L/\Lambda, M_e \le \xi \cdot L/\Lambda\}$ can be obtained by summing the probabilities of all legal arrangements (over all legal values of i, j, and k satisfying conditions 1, 2, and 3, as shown in the lemma. □

Based on $P\{M_s = \lambda \cdot L/\Lambda, M_e \le \xi \cdot L/\Lambda\}$ and $0 \le \lambda \le \xi$, we have:

$$P\{M_e \le \xi L/\Lambda\} = \sum_{\lambda=0}^{\xi} P\{M_s = \lambda L/\Lambda, M_e \le \xi L/\Lambda\}.$$

Hence,

$$P\{M_e = \xi L/\lambda\} = \begin{cases} P\{M_e \le 0\} & \text{if } \xi = 0 \\ P\{M_e \le \xi L/\Lambda\} - P\{M_e \le (\xi-1)L/\Lambda\} & \text{if } 1 \le \xi \le \Lambda. \end{cases}$$

Therefore, $\mathbf{E}(M_e)$ can be derived as:

$$\mathrm{E}(M_e) = \frac{L}{\Lambda} \cdot \sum_{\xi=0}^{\Lambda} (\xi \cdot P\{M_e = \xi L/\Lambda\}). \tag{3-3}$$

So far, we have solved Problem 3.1 or, more specifically, the expected extents L_1 and L_2 of the two new nodes after splitting L are:

$$L_1 = \mathbf{E}(M_e), \text{ and } L_2 = L - \mathbf{E}(I_{m+1}.s), \tag{3-4}$$

where $\mathbf{E}(I_{m+1}.s)$ and $\mathbf{E}(M_e)$ are represented in (3-2) and (3-3), respectively. It is clear that, for arbitrary object extents, the new nodes produced by the R* split algorithm do not necessarily have similar extents on the split dimension (actually as mentioned in the sequel, this is the case only if all the objects have the same sizes).

Now, we are ready to clarify the procedures of computing the ERF_i for the ith dimension, given the corresponding length distribution F_i. First, it is easy to see that $ERF_i(0) = 1$ because the extent of a root covers the entire universe[4] before any split is performed. $ERF_i(j+1)$ is obtained by taking the average extent after splitting a node with extent $ERF_i(j)$. Therefore,

4. In general, $ERF_i(0)$ equals the axis length of the ith dimension. Here, it is 1 because we assume unit data space.

Algorithm Compute_ERF_i (F_i, t)

/*F_i is the length distribution function of the ith dimension. The algorithm computes the values of the $ERF_i(0)$, $ERF_i(1)$, …, $ERF_i(t)$, where t is the second parameter (an integer)*/

1. $ERF_i(0)$=axis length at dimension i, and j=0
2. while ($j \le t$) do
3. $L=ERF_i(j)$
4. construct $F\{\xi\}$ as in equation 3-6
5. compute $\mathbf{E}(I_{m+l}.s)$ and $\mathbf{E}(M_e)$ as in equations 3-2 and 3-3 respectively
6. compute L_l and L_2 as in equation 3-4
7. $ERF_i(j)=\frac{1}{2}(L_l+L_l)$
8. $j=j+1$

Fig. 11. Computing ERF_i.

Algorithm Compute_L_{ij} (*ERFT*)

/*this function estimates L_{ij} i.e., the average extent of a level-i node along the jth dimension $0 \le i \le h-1$, $1 \le j \le d$, based on the *ERF* table *ERFT* specified */

1. compute h and N_i ($0 \le i \le h-1$) by equation 2-5
2. for each level i=0 to $h-1$
3. find the combination (s_{i1}, s_{i2}, …, s_{id}) that minimizes $\sum_{j=1\sim d} ERF_i(s_{ij})$ with the constrain $\sum_{j=1\sim d} s_{ij}=\log_2 N_i$
4. for each dimension j=1 to d
5. $L_{ij} = ERF_j(s_{ij})$

End Compute_L_{ij}

Fig. 12. Compute L_{ij}.

$$ERF_i(j+1) = 1/2 \cdot (L_1 + L_2), \tag{3-5}$$

where L_1 and L_2 are estimated from (3-4) by setting L to $ERF_i(j)$ and m to $\lfloor b/2 \rfloor$ (i.e., on average, each split distributes the entries evenly). It is worth mentioning that the probability density function F used in Problems 3.1 and 3.2 is obtained from the length distribution $F_i\{\xi\}$ (along the split dimension i). Specifically, $F\{\xi\}$ is the probability $P\{r.l_i \le \xi\}$ (that the extent length of data rectangle r is shorter than ξ on the ith axis) with the implicit condition $r.l_i \le L$, i.e., the extent must be completely in that (length L) of the node containing it. Hence, $F\{\xi\}$ equals

$$F(\xi) = P\{r.l_i \le \xi | r.l_i \le L\} = \frac{P\{r.l_i \le \xi\}}{P\{r.l_i \le\}} = \frac{F_i(\xi)}{F_i(L)}. \tag{3-6}$$

Fig. 11 summarizes the procedures for computing the values of ERF_i.[5]

3.3 Estimating MBR Extents L_{ij}

The ERF_i along each dimension is a discrete function such that $ERF_i(s_{ij})$ (where s_{ij} is an integer) gives the extent of the level-j node (along the ith dimension) after performing s_{ij} splits. Note that, in practice, the value of s_{ij} is usually small. For example, if $s_{i1} = s_{i2} = 20$, then s_i (the total number of splits performed on a node at level i) equals 40 (since $s_i \le s_{ij}$). By Observation 2, this implies that the total number N_i of level-i nodes should be more than 2^{40}. Given that a typical fanout of a node (i.e., average number of entries in a node) is more than 50, the database should contain more than 500 trillion records!

Thus, for typical databases, it suffices to precompute the values of $ERF_i(s_i)$ for a small range of values for s_i (we suggest $[0, 20]$, i.e., invoke the algorithm of Fig. 11 by setting t to 20). The precomputation overhead is small and is an one-time effort (i.e., the results can be used for all future queries). The computed values are stored in an *ERF table* where $ERFT[i][j](1 \le i \le d, 0 \le j \le 20)$ gives the value of $ERF_i(j)$ (i.e., the extent of a node on the ith dimension after j splits on this axis). With $ERFT$, we can estimate L_{ij} (i.e., the average extent of a level-i node along the jth dimension) with a few table lookups to solve the constrained optimization problem as stated in Observation 4. In the sequel, we illustrate this in 2-dimensional space as the discussion extends to general d-dimensional space trivially. For simplicity, we first assume that N_i (estimated in (2-5)) is a power of 2 (i.e., $N_i = 2^k$ for some integer k).

Let s_{01} and s_{02} be the number of splits performed at the leaf level on the first and second dimensions, respectively. As indicated in Observation 4, we can find, among the combinations of (s_{01}, s_{02}) that satisfy $s_{01} + s_{02} = \log_2(N_0)$ (review Observation 2), the pair that minimizes $ERF_1(s_{01}) + ERF_2(s_{02})$. For instance, let $s_{01} + s_{02} = 15$; then, the combinations of (s_{01}, s_{02}) include $\{(0, 15), (1, 14), \ldots, (14, 1), (15, 0)\}$ (for each combination, the value of $ERF_i(s_{0i})$ can be found in $ERFT[i][s_{0i}]$). This approach generalizes to higher levels easily with the only exception that $s_i = \log_2(N_i)$. Such operations are cheap (due to the small number of combinations) and must be performed only once. Fig. 12 summarizes the steps for computing L_{ij}.

5. Equation (3-2) requires numerical evaluation. We adopt the trapezoidal rule that evaluates a general one-layer integral $\int_a^b f(x)dx$ by calculating the values $f(x_i)$ at regular positions $x_i = a + i \cdot (b-a)/c (0 \le i \le c)$, where c is a constant (100 in our experiments) of the integral range $[a, b]$. Then, the integral value can be approximated as $\frac{b-a}{2c} \sum_{i=0}^{c-1} [f(x_i) + f(x_{i+1})]$.

Next, we remove the constraint that N_i must be a power of 2. In this case, some leaf nodes (let their number be n_1) have split $s_0(=\lfloor \log_2(N)0) \rfloor)$ times, while other leaf nodes (let their number be n_2) have split $s_0\prime(=\lceil \log_2(N_0) \rceil)$ times. As suggested in [8], $n_1 = 2^{s_0+1} - N_0$ and $n_2 = 2N_0 - 2^{s_0'}$. For the $n_1(n_2)$ nodes that split $s_0(s_0')$ times, their extents $l_{1i}(l_{2i})$ along each dimension i can be obtained using the algorithm in Fig. 12. Then, the final L_{0i} is estimated as the weighted sum of l_{1i} and l_{2i}:

$$L_{0i} = \frac{n_1}{N_0} l_{1i} + \frac{n_2}{N_0} l_{2i}.$$

Extending to higher levels is trivial (just replacing N_0 with N_i). We emphasize that L_{ij} for all levels and dimensions can be precomputed offline, and the time to produce a cost estimate in query optimization involves only the cost of evaluating (2-4) with the specified query parameters (i.e., in the order of milliseconds). Finally, capturing the performance of nonuniform data is relatively easy by using a histogram as introduced in Section 2. Specifically, for a query inside a histogram cell with N_c objects, the query cost is estimated based on a conceived uniform data set with $N_c \cdot H^2$ objects.

4 Applications of *ERF*s

The first step in applying the *ERF* technique is to obtain the data length distribution on each dimension. In particular, it has been shown that such distributions, in several contexts, can be described using mathematical models (e.g., the *regal law* for spatial data [30]), while in other cases, the distribution can always be approximated from the length statistics of a random sample. Section 4.1 shows that conventional spatial data sets constitute a simple instance of the *ERF* application and explains, for the first time, the reason why even though most real data sets contain objects with various sizes, their performance can still be accurately predicted by assuming that objects have the same extents. Section 4.2 focuses on temporal data (including spatio-temporal and multimedia objects), where existing analytical methods fail, and explicitly quantifies the factors that lead to this failure. Section 4.3 comments on the applicability of *ERF*s to other access methods.

4.1 Spatial Data

We first derive the *ERFs* for the special case that all data objects have the same extent. Specifically, each d-dimensional rectangle has length L_{Di} along the ith dimension $(1 \le i \le d)$, or in other words, the length distribution for dimension $i(1 \le i \le d)$ is step-wise: $F_i(\xi) = 0$ (if $\xi < L_{Di}$) and 1 (if $\xi \ge L_{Di}$). ERF_i can be computed by the algorithm in Fig. 11, except that (3-3) (for $\mathbf{E}(M_e)$ as in line 5 of the algorithm) can be significantly simplified. This is because, unlike the general case of Problem 3.1, the sorting of $I_1, I_2, \ldots, I_b$ by their starting points also implies the order of their ending points, namely, given two intervals I_j and I_k, $I_j.s \le I_k.s$ if and only if $I_j.e \le I_k.e$. As a result, we have $\mathbf{E}(M_e) = \mathbf{E}(I_m.s + L_{Di}) = \mathbf{E}(I_m.s) + L_{Di}$. Applying a derivation similar to $\mathbf{E}(I_{m+1}.s)$ (see Section 3.2), we have

$$\mathrm{P}\{I_m.s \le \xi\} = \sum_{i=m}^{b} \binom{m}{i} (\mathrm{P}\{I.s \le \xi\})^i (1 - \mathrm{P}\{I.s \le \xi\})^{b-i},$$

$$\mathrm{E}(I_m.s) = \int_0^L \xi \cdot \frac{d\mathrm{P}\{I_m.s \le \xi\}}{d\xi} d\xi, \tag{4-1}$$

where $P\{I.s \le \xi\}$ is given in Lemma 3.1. Solving (3-2), (4-1), and (3-4), we obtain $L_1 \approx L_2 \approx 1/2(L + L_{Di})$. Hence, by (3-5), we obtain ERF_i as:

$$ERF_i(j+1) \approx 1/2 \cdot (ERF_i(j) + L_{Di}). \tag{4-2}$$

When L_{Di} is small, (4-2) becomes $ERF_i(j) \approx 2^{-j}$, which, interestingly, is exactly the "coarse" form of the *TS* model presented in [3]. Bohm [3] also shows that the difference from the actual model is negligible in low-dimensional spaces, meaning that, in this case, *ERFs* basically degenerate to *TS*. This equation has another important implication: *for small L_{Di}, object extents do not affect the node extents, in which case, the performance of R*-trees can be predicted by simply treating (rectangular) objects as points*. This property is satisfied in many real spatial data sets (containing objects with variable but small sizes) and, therefore, the *TS* model, which assumes that objects have similar sizes, can still produce satisfactory estimation (with histograms).

Saltenis and Jensen [38] observe that scaling the dimensions of the universe can considerably improve query performance of R-trees, which can also be explained with *ERFs*. Stepping even further, in the sequel, we show how to use *ERFs* to determine the optimal scaling that minimizes the query cost. Consider, for example, a 2D uniform data set with N points. Assume a query with extents q_1 and q_2 along the x and y-dimensions, and that the average extents of a leaf node are L_{01} and L_{02}, respectively. The access probability for a leaf node (2-1) can be rewritten as: $q_1 \cdot q_2 + q_1 \cdot L_{02} + q_2 \cdot L_{01} + L_{01} \cdot L_{02}$. Obviously, $q_1 \cdot q_2$ is not affected by node extents. Now, assume that a leaf node splits s_{01} and s_{02} times along the x and y-dimensions (where $s_{01} + s_{02} = \log_2 N/f$) respectively; then, by (4-2), we have $L_{01} = 1/2^{S_{01}}$ and $L_{02} = 1/2^{S_{02}}$. Thus, it follows that $L_{01} \cdot L_{02} = 1/2^{S_{01}+S_{02}} = f/N$, which is also a constant. Therefore, minimizing the access probability is equivalent to minimizing $q_1 \cdot L_{02} + q_2 \cdot L_{01}$, resulting in $q_1/q_2 = L_{01}/L_{02}$, which implies that *the optimal node extent should have a similar aspect ratio to that of the query*. Assuming $q_1/q_2 = 2^\xi > 1$ (for simplicity, consider ξ to be an integer), the optimal split times s_{01} and s_{02} should satisfy $s_{02} = s_{01} - \xi$. Notice that this can be achieved by simply scaling up the y-dimension 2^ξ times, which renders the ERF_y of y-dimension to be: $ERF_y(j) = 2^{\xi - j}$. Given that ERF_x remains the same (i.e., $ERF_x(j) = 2^{-j}$), it is easy to verify that the algorithm of Fig. 12 indeed leads to the optimal node extents.

4.2 (Spatio)Temporal and Multimedia Data

Temporal data (including spatio-temporal and multimedia objects) consist of 1) a temporal and 2) at least one nontemporal dimension (e.g., salary, positions, as in Figs. 2a and 2b). The *ERFs* of the nontemporal dimensions are the

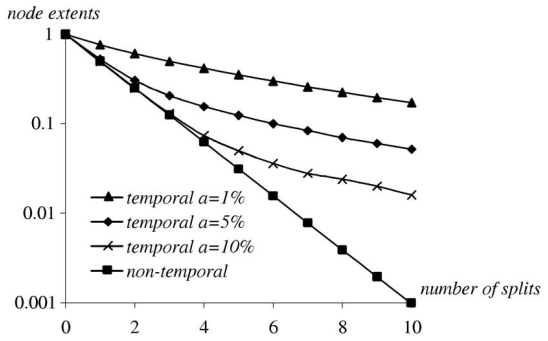

Fig. 13. *ERF* values for temporal and nontemporal dimensions.

same as those for conventional spatial objects (i.e., (4-2)). As discussed in [45], if T is the length of the history (in terms of number of recorded timestamps) and ξ a positive integer smaller than T, then the probability that an object's lifespan L_t (i.e., its temporal extent length) is smaller than ξ is represented as (i.e., the temporal length distribution):

$$F_t(\xi) = P\{L_t \leq \xi/T\} = \sum_{i=1}^{\xi} \left[a(1-a)^{i-1}\right], \qquad (4\text{-}3)$$

where the *agility a* is a constant between $[0, 1]$ describing the percentage of records that issue updates per timestamp. A large agility indicates that objects' attributes change with high frequency (e.g., vehicle locations have higher agilities than account balances as they are updated more often).

It has been observed [22], [7], [40] that the node extents of R-trees indexing temporal data are usually much longer on the time dimension than the other (nontemporal) dimensions. To explain this, Fig. 13 shows the computed *ERF* values (as a function of split times, obtained from the algorithm in Fig. 11) for temporal (agility a = 1 percent, 5 percent, 20 percent, respectively, based on (4-3)) and nontemporal (based on (4-2)) dimensions.

It is clear that the lifespan of a node decreases much more slowly (after successive splits) than the nontemporal dimension, implying that a split on the temporal dimensions is less effective, which leads to significant overlap between the two new nodes (after the split). Note that, when the agility increases, the lifespans of objects become shorter and, hence, the node extents decrease faster. In the extreme case where the agility is 100 percent (i.e., each object remains alive for a single timestamp), the temporal *ERF* equals the nontemporal one.

In order to quantify the node extents, consider a data set with a nontemporal and a temporal (agility 5 percent) dimension. Assume a leaf node totally splits 12 times; then, passing the *ERF* values in Fig. 13 to the algorithm in Fig. 12, we decide that optimally a node should split six times on both the temporal and nontemporal dimensions. According to Fig. 13, the resulting extents would be in the order of 0.1 and 0.01 on the two dimensions, respectively (i.e., about 10 times longer on the time dimension). Further, these values, when applied to (2-4) together with the query parameters, produce estimates for the corresponding query cost.

4.3 Application of *ERF*s to Other Structures

Although this paper focuses on R*-trees, the concept of extent regression functions also applies to other structures, as long as 1) Observation 3 (i.e., the min-marg rule) holds (which essentially implies that the splitting of this structure is "effective"), and 2) each node split is performed on one dimension so that the extents on the other axes are not affected. Access methods satisfying these conditions include the X-[8], A-[41], and KDB-trees [28], etc. What distinguishes each case is the derivation of the corresponding *ERF* based on the concrete split characteristics (in the same way as we did for R*-trees in Section 3).

In some structures (such as the original R-tree), however, a single split may affect the node extents on all dimensions (e.g., by the linear or quadratic split algorithm [14]), in which case the *ERF* should represent the node extent as a function of the *total* number of splits ever performed on this node (nevertheless, for each axis, there is still a specialized *ERF*, based on the length distribution of objects on that dimension). The difference from the above one-axis-split structures is that, here the problem is no longer an optimization issue as defined in Observation 4. The concrete derivation, as mentioned earlier, requires a careful investigation of the split algorithms, which falls out of the scope of this paper (and is less interesting due to the inferior performance of the original R-tree).

5 Experiments

In this section, we experimentally evaluate the effectiveness of the proposed method using a Pentium III 700 MHz CPU with 256 Mbytes memory. The disk page is set to 1,024 bytes in all cases,[6] such that the node capacity of R*-trees equals 48 for 2-dimensional, and 30 for 3-dimensional spaces. We compare our method (*ERF*) with the three existing approaches discussed in Section 2, namely,

1. the *TS* model [46],
2. the *fractal* method [20], and
3. a hypothetical *selectivity* method, which predicts the cost of a window query using the formula $\sum_{i=0\sim(h-1)} \lceil n/f^{i+1} \rceil$, where n is the *actual* number of objects satisfying the query, f the average node fanout, and h the height of the tree.

Note that, in practice, a selectivity prediction technique would introduce additional error in the estimation of n and, hence, would be less accurate than the presented results. For each method, we measure its average relative error in answering a workload of 200 queries with the same parameters. If est_i and act_i is the estimated and actual costs of the ith query $(1 \leq i \leq 200)$, respectively, then the average relative error is computed as

$$(1/200) \cdot \sum_{i=1\sim200} |est - act|/act.$$

6. A relatively small page size combined with low data set cardinality can also capture performance in practical situations where both values are significantly higher.

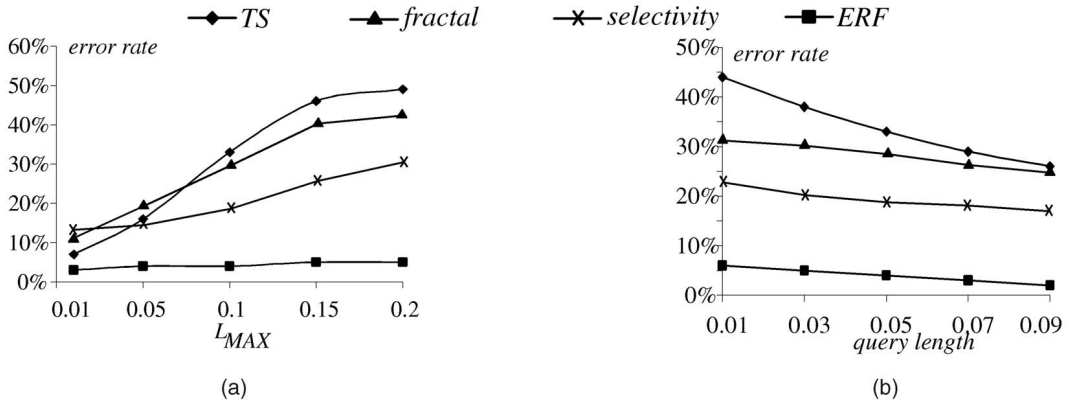

Fig. 14. Error for uniform length distributions (spatial data). (a) Error versus maximum data length $L_{MAX}(qs = 0.05)$. (b) Error versus query length $qs(L_{MAX} = 0.1)$.

5.1 Evaluation for Spatial Datasets

This section deals with two-dimensional data whose attributes do not evolve with time (i.e., the context where the previous models are derived). All the axes of the data space have unit length. Each data set[7] contains 100k rectangles such that 1) their locations uniformly distribute in the data space and 2) the lengths of their extents follow uniform or exponential distributions (the extent on each axis is generated independently). For uniform (length) distributions, the extent length distributes uniformly in the range $[0, L_{MAX}]$, where L_{MAX} is a data set constant. Exponential distributions are decided by the regal law [30], or specifically, the probability that the area of a data MBR is greater than a certain value ξ equals $A^{-\xi}$, where A is a constant. However, unlike [30], data rectangles are not required to have a fixed aspect ratio; instead, the lengths of each node's MBR on the two-axes are independently generated.

Each query 1) is a square with the same extent length qs (a workload parameter) on each dimension and 2) distributes uniformly in the data space. To apply TS, we need the data set density (i.e., the parameter D in (2-3)), which is calculated as the total area of all the data rectangles divided by that of the data space (this is equivalent to using the average data size to model all objects). For *fractal*, since there does not exist a general technique that works for all length distributions and types of objects (i.e., points and rectangles), we evaluate two alternatives and report the better result in each case. Specifically, the first method adopts the regal law [30], while the other one ignores the object extents and employs the method of [12] for point data (as suggested in [1]). In both cases, we measure the fractal dimension of a rectangle data set by using the object centroids (the same approach was followed in [1]). In the following experiments, the model in [12] outperforms that in [30] in most cases. Hence, we report the performance according to [12], unless specifically stated.

Fig. 14 shows the results for uniform length distributions. Specifically, in Fig. 14a, we fix the query extent $qs = 0.05$ and vary L_{MAX} from 0.01 to 0.2 (up to 20 percent of the data axis length). *ERF* achieves high accuracy for all L_{MAX} (maximum error 6 percent). The performance of the other methods deteriorates significantly as L_{MAX} increases due to different reasons. TS fails because the average size cannot capture the large extent variance that occurs in case of high L_{MAX}. *Fractal*, optimized for point data, performs well only for small rectangles (i.e., low L_{MAX}), so that object extents do not bias the actual query costs considerably. *Selectivity* is accurate only if most leaf nodes contribute as many qualifying data objects as the fanout, meaning that the MBRs of these nodes must fall inside the query window completely. A large value of L_{MAX} increases the node MBRs and decreases the probability for a node to fall inside the query.

Fig. 14b shows the results when fixing L_{MAX} to the median value 0.1 and varying the query extent qs from 0.01 to 0.09. Note that all methods improve with the query size. This is expected because, for larger queries, 1) the query size plays a more important role (than the node extents) in the probability that a node is accessed (explaining the improvement of *TS*, *fractal*, *ERF*) and 2) more nodes fall completely in the query window (explanation for *selectivity*). *ERF* is again very accurate (maximum error around 7 percent) and outperforms its competitors significantly in all cases.

Fig. 15 shows the results of similar experiments for exponential length distributions. Fig. 15a plots the error rates as a function of A (base of the exponential probability function), for $qs = 0.05$. *ERF* is highly accurate for all values of A (below 5 percent error), while the other methods are erroneous for small A, but improve as A increases. The explanation is similar to that for Fig. 14a, given the fact that a low value of A also leads to objects with small sizes. Fig. 15b sets A to 4 and illustrates the error rate as a function of the query size (the results of this experiment are obtained using the model in [30]). As with Fig. 14b, the error of all methods decreases for larger queries.

Recall that, to apply *ERF*, we first need to invoke the algorithm of Fig. 12 in order to determine the R-tree node extents (based on objects' length distribution). Table 2 shows the execution time of this algorithm for the above data sets. The processing time is less than two minutes in all cases, and is only affected by the type of length distribution

7. We also experimented with "conventional" spatial data [43], where the object extentes are very small compared to the data space. The results are omitted because, as analyzed in Section 4.1, in this case, *ERF* essentially degenerates into *TS*, whose performance has been extensively evaluated in [46], [47].

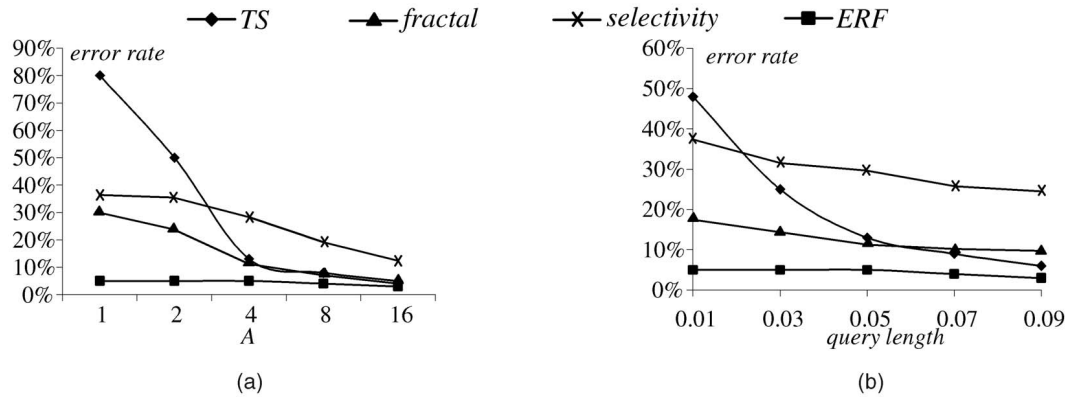

Fig. 15. Error for exponential length distributions (spatial data). (a) Error versus the base of exp. function $A(qs = 0.05)$. (b) Error versus query length $qs(A = 4)$.

TABLE 2
Offline Preprocessing Time of *ERF* (Spatial Data)

Uniform length distribution		Exponential length distribution	
L_{MAX}	Processing time (seconds)	A	Processing time (seconds)
0.01	69	1	92
0.05	70	2	93
0.1	71	4	93
0.15	70	8	92
0.2	72	16	95

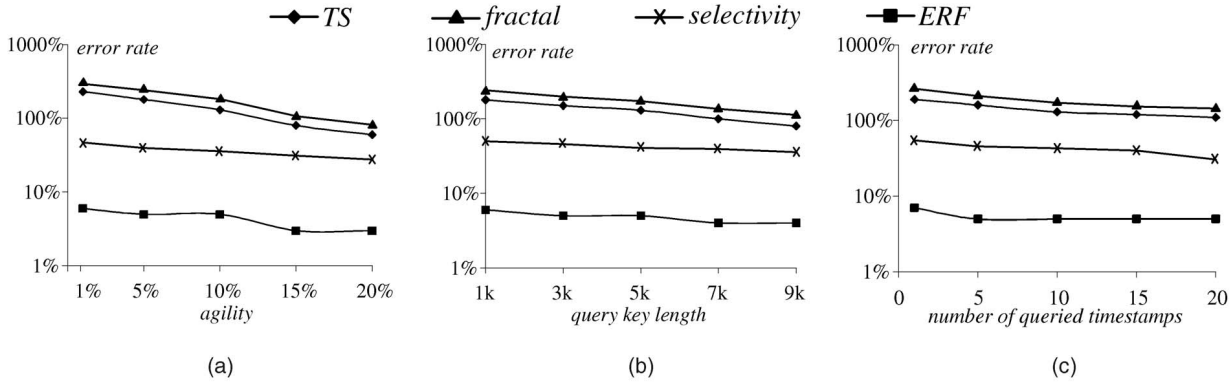

Fig. 16. Error for temporal (two-dimensional) data sets. (a) Error versus $a(qs = 0.05, qt = 10)$. (b) Error versus $qs(a = 10\%, qt = 10)$. (c) Error versus $qt(a = 10\%, qs = 0.05)$.

(i.e., it is independent of the concrete parameters of the distributions). As discussed in Section 3.3, this processing can be done completely offline, so that the actual overhead to produce an estimate involves only that of evaluating (2-4) (which is negligible).

5.2 Evaluation for Temporal and Spatio-Temporal Data Sets

Having demonstrated the superiority of *ERF* for spatial data, the remaining experiments focus on temporal and spatio-temporal data sets, indexed by 2D and 3D R-trees, respectively. For the temporal case, we generate time-evolving data as follows: At the first timestamp, values of 100k search keys are uniformly generated in the range $[0, 100000]$. Then, at each of the consecutive 1,000 timestamps, a percent (i.e., the data set *agility*) of the objects are selected to produce random changes. Accordingly, a query has two parameters: length qs along the search key dimension and the number qt of timestamps involved. Each workload consists of queries with the same parameters such that, the starting point of each query's key (temporal) range is generated uniformly in $[0, 100000 - qs]([0, 1000 - qt])$. As with Section 5.1, we compare *ERF* to *TS*, *fractal*, and *selectivity*.

Fig. 16a shows the error rate as a function of data set agility, setting the parameters $qs = 0.05$ and $qt = 10$ (timestamps). In Figs. 16b and 16c, the agility is fixed to 10 percent and all the methods are tested with respect to different qs (fixing $qt = 10$) and qt $(qs = 0.05)$, respectively. Evidently, the previous solutions completely fail to capture the query costs (note that the data records are 2D intervals, parallel to the time axis, with vastly different lengths). On the other hand, *ERF* performs very well in all settings, yielding maximum error 9 percent.

In order to generate spatio-temporal data, we follow the approach taken in [44]. Specifically, two real data sets

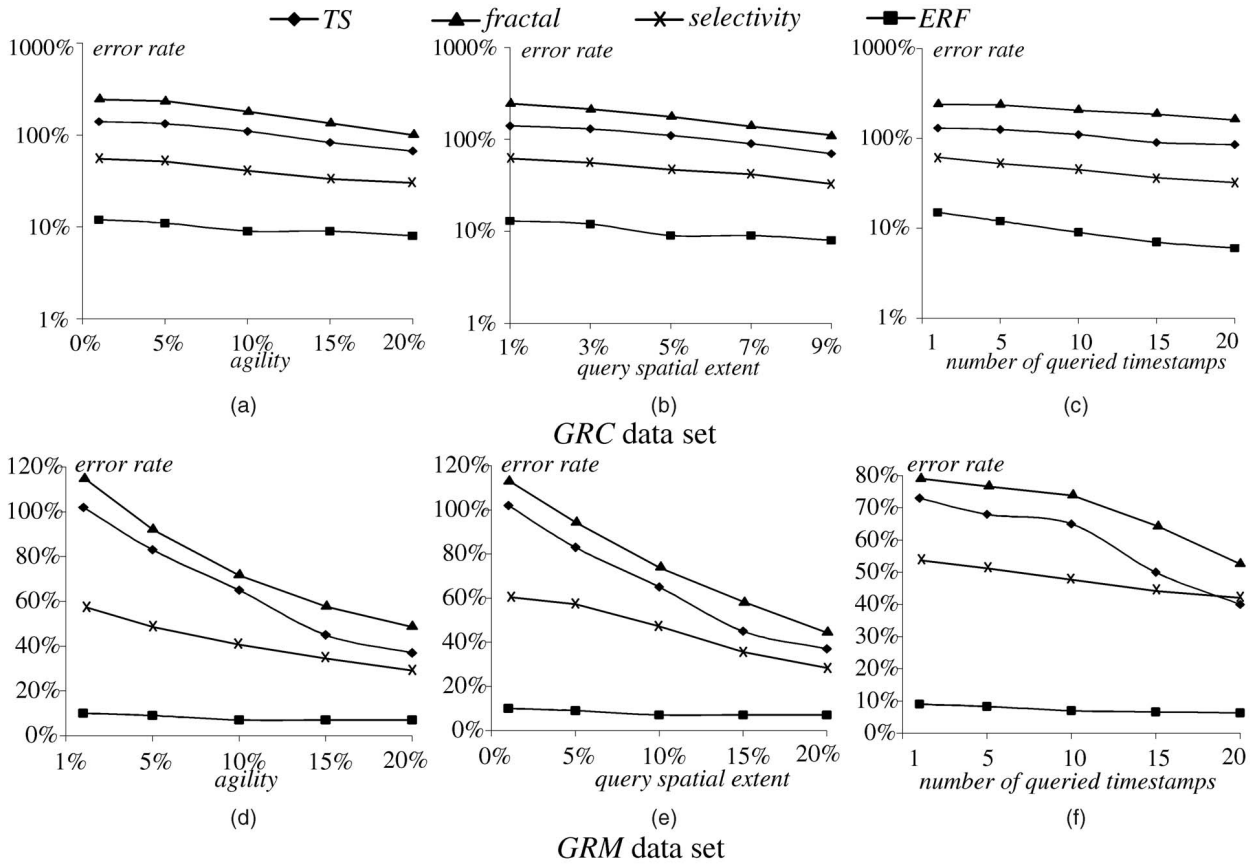

Fig. 17. Error for spatio-temporal (3D) data sets. (a) Error versus $a(qs = 0.05, qt = 10)$ for both *GRC* and *GRM* data sets. (b) Error versus $qs(a = 10\%, qt = 10)$ for both *GRC* and *GRM* data sets. (c) Error versus $qt(a = 10\%, qs = 0.05)$ for both *GRC* and *GRM* data sets.

constitute the initial location distribution at the first timestamp: 1) *GRC*, containing poly-lines that represent 23k road segments in Greece [50], and 2) *GRM*, containing 18k rectangles that represent utility network elements in Germany [50]. Then, at each of the following 1,000 timestamps, a percent of the objects are selected to produce random position changes. It is worth mentioning that, data sets generated this way incur (slow) location distribution changes (to uniformity) as time evolves. For both *TS* and *ERF*, we maintain a 3D histogram (a generalized version of the one described in Fig. 5) with $H = 30$ (i.e., each dimension is divided into 30 partitions). Similar to the temporal case, each query has two parameters: 1) the spatial extent qs (identical along the two spatial dimensions, denoted as the percentage of the universe axis) and 2) the temporal length qt (number of timestamps queried). Queries' spatial (temporal) ranges in a workload are uniformly generated on the corresponding axes. Fig. 17 demonstrates the results for *GRC* and *GRM*.

Figs. 17a and 17d plot the error rate as a function of agility, using query workloads with $qs = 0.05$ and $qt = 10$. For Figs. 17b and 17e, $a = 10\%$, $qt = 10$, and qs varies in the range $[0.01, 0.09]$. In Fig. 17c and 17f, a and qs are set to their median values (10 percent and 0.05, respectively), while the number of queried timestamps ranges between 1 and 20. The results are consistent with that of temporal data (Fig. 16). In all cases, our approach captures the R-tree behavior very well, yielding errors around 10 percent, while the other techniques again fail to predict performance.

Similar to Table 2, Table 3 shows the preprocessing time of *ERF* (i.e., executing the algorithm in Fig. 12) as a function of the agility, for the temporal and spatio-temporal data sets examined. The overhead is the same for all data sets because their records have the same length distributions (on both temporal and nontemporal dimensions).

6 Conclusion

In this paper, we introduce the notion of the extent regression function and develop a novel analytical framework that is considerably more powerful than the previous R-tree cost models. The merit of our technique is that it can be easily applied for query optimization in several popular traditional (e.g., temporal, multimedia) and emerging (e.g., spatio-temporal) database systems, where the existing methods completely fail. Further, we not only present the

TABLE 3
Offline Preprocessing Time of *ERF*
(Spatial, Spatio-Temporal Data)

Temporal, and spatio-temporal (*GRC* and *GRM*)	
Agility	Processing time (seconds)
1%	75
5%	76
10%	74
15%	74
20%	75

theory, but also demonstrate the application of *ERFs* in the aforementioned scenarios, and obtain several novel insights into the R-tree performance. Extensive experiments prove that the proposed method provides accurate estimations in all settings.

For the special case where the object extents are very small compared to the data space, our model becomes equivalent to that of [46], which has been widely used for conventional R-tree applications (i.e., spatial data). Although not rigorously proven in the paper, we conjecture that, if the data lengths follow exponential distribution (i.e., satisfying the regal law) and data have fixed aspect ratio, *ERFs* should degenerate into the results in [30]. However, unlike [30], which assumes that the extents of a node are similar on all axes, our technique leads to a different *ERF* for each axis, depending on the corresponding data length distribution on that dimension.

Future work may focus on the refinement of *ERFs* under various settings (e.g., in the presence of buffers [24]), and their application in query optimization. For example, as discussed in Section 4, *ERFs* can be used to decide the optimal node extents of an index with respect to specific query parameters and, thus, may possibly lead to the development of new workload-aware indexes.

ACKNOWLEDGMENTS

This work was supported by HKUST 6180/03E from Hong Kong RGC.

REFERENCES

[1] S. Acharya, V. Poosala, and S. Ramaswamy, "Selectivity Estimation in Spatial Databases," *Proc. ACM SIGMOD Int'l Conf. Management of Data,* 1999.

[2] A. Aboulnaga and J. Naughton, "Accurate Estimation of the Cost of Spatial Selections," *Proc. Int'l Conf. Data Eng.,* 2000.

[3] C. Bohm, "A Cost Model for Query Processing in High Dimensional Data Spaces," *ACM Trans. Database Systems,* vol. 25, no. 2, pp. 129-178, 2000.

[4] B. Babcock, S. Chaudhuri, and G. Das, "Dynamic Sample Selection for Approximate Query Processing," *Proc. 2003 ACM SIGMOD Int'l Conf. Management of Data,* 2003.

[5] A. Belussi and C. Faloutsos, "Estimating the Selectivity of Spatial Queries Using the Correlation's Fractal Dimension," *Proc. 21st Int'l Conf. Very Large Data Bases (VLDB),* 1995.

[6] N. Bruno, L. Gravano, and S. Chaudhuri, "STHoles: A Workload Aware Multidimensional Histogram," *Proc. ACM SIGMOD Int'l Conf. Management of Data,* 2001.

[7] R. Bliujute, C. Jensen, S. Saltenis, and G. Slivinskas, "R-Tree Based Indexing of Now-Relative Bitemporal Data," *Proc. 24th Int'l Conf. Very Large Data Bases,* 1998.

[8] S. Berchtold, D. Keim, and H.P. Kriegel, "The X-Tree: An Index Structure for High-Dimensional Data," *Proc. Int'l Conf. Very Large Databases (VLDB),* 1996.

[9] B. Blohsfeld, D. Korus, and B. Seeger, "A Comparison of Selectivity Estimators for Range Queries on Metric Attributes," *Proc. Int'l Conf. Very Large Databases (VLDB),* 1999.

[10] N. Beckmann, H. Kriegel, R. Schneider, and B. Seeger, "The R*-Tree: An Efficient and Robust Access Method for Points and Rectangles," *Proc. ACM SIGMOD Int'l Conf. Management of Data,* 1990.

[11] A. Deshpande, M. Garofalakis, and R. Rastogi, "Independence Is Good: Dependency-Based Histogram Synopses for High-Dimensional Data," *Proc. 2001 ACM SIGMOD Int'l Conf. Management of Data,* 2001.

[12] C. Faloutsos and I. Kamel, "Beyond Uniformity and Independence, Analysis of R-Trees Using the Concept of Fractal Dimension," *Proc. ACM SIGACT-SIGMOD-SIGART Principles of Database Systems,* 1994.

[13] C. Faloutsos, T. Sellis, and N. Roussopoulos, "Analysis of Object Oriented Spatial Access Methods," *Proc. ACM SIGMOD Int'l Conf. on Management of Data,* 1987.

[14] A. Guttman, "R-Trees: A Dynamic Index Structure for Spatial Searching," *Proc. ACM SIGMOD Int'l Conf. Management of Data,* 1984.

[15] D. Gunopulos, G. Kollios, V. Tsotras, and C. Domeniconi, "Approximate Multi-Dimensional Aggregate Range Queries over Real Attributes," *Proc. 2000 ACM SIGMOD Int'l Conf. Management of Data,* 2000.

[16] C. Jermaine, "Making Sampling Robust with APA," *Proc. VLDB Conf.,* 2003.

[17] J. Jin, N. An, and A. Sivasubramaniam, "Analyzing Range Queries on Spatial Data," *Proc. Int'l Conf. Data Eng.,* 2000.

[18] M. Jurgens and H. Lenz, "PISA: Performance Models for Index Structures with and without Aggregated Data," *Proc. 11th Int'l Conf. Scientific and Statistical Database Management,* 1999.

[19] I. Kamel and C. Faloutsos, "On Packing R-Trees," *Proc. Second Int'l Conf. Information and Knowledge Management (CIKM),* 1993.

[20] I. Kamel and C. Faloutsos, "Hilbert R-Tree: An Improved R-Tree Using Fractals," *Proc. 20th Int'l Conf. Very Large Databases,* 1994.

[21] C. Kolovson and M. Stonebraker, "Indexing Techniques for Historical Databases," *Proc. Int'l Conf. Data Eng.,* 1989.

[22] C. Kolovson and M. Stonebraker, "Segment Indexes: Dynamic Indexing Techniques for Multi-Dimensional Interval Data," *Proc. ACM SIGMOD Int'l Conf. Management of Data,* 1991.

[23] J. Lee, D. Kim, and C. Chung, "Multidimensional Selectivity Estimation Using Compressed Histogram Information," *Proc. 1999 ACM SIGMOD Int'l Conf. Management of Data,* 1999.

[24] S.T. Leutenegger and M.A. Lopez, "The Effect of Buffering on the Performance of R-Trees," *IEEE Trans. Knowledge and Data Eng.,* vol. 12, no. 1, pp. 33-44, Jan./Feb. 2000.

[25] X. Lin, Q. Liu, Y. Yuan, and X. Zhou, "Multiscale Histograms: Summarizing Topological Relations in Large Spatial Datasets," *Proc. 29th Int'l Conf. Very Large Data Bases,* 2003.

[26] Y. Mattias, J. Vitter, and M. Wang, "Dynamic Maintenance of Wavelet-Based Histograms," *Proc. 26th Int'l Conf. Very Large Data Bases,* 2000.

[27] Y. Mattias, J. Vitter, and M. Wang, "Wavelet-Based Histograms for Selectivity Estimation," *Proc. 1998 ACM SIGMOD Int'l Conf. Management of Data,* 1998.

[28] O. Procopiuc, P. Agarwal, L. Arge, and J. Vitter, "Bkd-Tree: A Dynamic Scalable kd-Tree," *Proc. Eighth Int'l Symp. Spatial and Temporal Databases,* 2003.

[29] G. Proietti and C. Faloutsos, "Accurate Modeling of Region Data," *IEEE Trans. Knowledge and Data Eng.,* vol. 13, no. 6, pp. 874-383, Nov./Dec. 2001.

[30] G. Proietti and C. Faloutsos, "I/O Complexity for Range Queries on Region Data Stored Using an R-Tree," *Proc. Int'l Conf. Data Eng.,* 1999.

[31] Y. Poosala and Y. Ioannidis, "Selectivity Estimation without the Attribute Value Independence Assumption," *Proc. 23rd Int'l Conf. on Very Large Data Bases,* 1997.

[32] D. Pfoser, C. Jensen, and Y. Theodoridis, "Novel Approaches to the Indexing of Moving Object Trajectories," *Proc. 26th Int'l Conf. Very Large Databases,* 2000.

[33] B.U. Pagel and H.W. Six, "Are Window Queries Representative for Arbitrary Range Queries?" *Proc. 15th ACM SIGACT-SIGMOD-SIGART Symp. Principles of Database Systems* 1996.

[34] B.U. Pagel, H.W. Six, H. Toben, and P. Widmayer, "Towards an Analysis of Range Query Performance in Spatial Data Structures," *Proc. 12th ACM SIGACT-SIGMOD-SIGART Symp. Principles of Database Systems,* 1993.

[35] D. Papadias, Y. Tao, P. Kalnis, and J. Zhang, "Indexing Spatio-Temporal Data Warehouses," *Proc. Int'l Conf. Data Eng.,* 2002.

[36] M. Stonebraker, "The Design of Postgres Storage System," *Proc. 13th Conf. Very Large Databases,* 1987.

[37] C. Sun, D. Agrawal, and A. El Abbadi, "Selectivity Estimation for Spatial Joins with Geometric Selections," *Proc. Conf. Extending Database Technology,* 2002.

[38] S. Saltenis and C. Jensen, "Indexing the Positions of Continuously Moving Objects," *The VLDB J.,* vol. 11, no. 1, pp. 1-16, 2002.

[39] T. Sellis, N. Roussopoulos, and C. Faloutsos, "The R+-Tree: A Dynamic Index for Multi-Dimensional Objects," *Proc. 13th Conf. Very Large Databases,* 1987.

[40] B. Salzberg and V. Tsotras, "Comparison of Access Methods for Temporal Data," *ACM Computing Surveys,* vol. 31, no. 2, pp. 158-221, 1999.
[41] Y. Sakurai, M. Yoshikawa, S. Uemura, and H. Kojima, "The A-Tree: An Index Structure for High-Dimensional Spaces Using Relative Approximation," *Proc. 26th Int'l Conf. Very Large Data Bases,* 2000.
[42] N. Thaper, S. Guha, P. Indyk, and N. Koudas, "Dynamic Multidimensional Histograms," *Proc. 2002 ACM SIGMOD Int'l Conf. Management of Data ,* 2002.
[43] http://www.census.gov/geo/www/tiger/, 2004.
[44] Y. Tao and D. Papadias, "Spatial Queries in Dynamic Environments," *ACM Trans. Database Systems,* vol. 28, no. 2, pp. 101-139, 2003.
[45] Y. Tao, D. Papadias, and J. Zhang, "Cost Models for Overlapping and Multi-Version Structures," *ACM Trans. Database Systems,* vol. 27, no. 3, pp. 299-342, 2002.
[46] Y. Theodoridis and T. Sellis, "A Model for the Prediction of R-Tree Performance," *Proc. 15th ACM SIGACT-SIGMOD-SIGART Symp. Principles of Database Systems,* 1996.
[47] Y. Theodoridis, E. Stefanakis, and T. Sellis, "Efficient Cost Models for Spatial Queries Using R-Trees," *IEEE Trans. Knowledge and Data Eng.,* vol. 12, no. 1, pp. 19-32, Jan./Feb. 2000.
[48] M. Vazirgiannis, Y. Theodoridis, and T. Sellis, "Spatio-Temporal Composition and Indexing for Large Multimedia Applications," *ACM/Springer Multimedia J.,* vol. 6, no. 4, 1998.
[49] Y. Wu, D. Agrawal, and A. Abbadi, "Applying the Golden Rule of Sampling for Query Estimation," *Proc. 2001 ACM SIGMOD Int'l Conf. Management of Data,* 2001.
[50] http://www.rtreeportal.org/, 2004.
[51] M. Wang, J. Vitter, L. Lim, and S. Padmanabhan, "Wavelet-Based Cost Estimation for Spatial Queries," *Proc. Seventh Int'l Symp. Spatial and Temporal Databases,* 2001.
[52] S. Yao, "Random 2-3 Trees," *Acta Informatica,* vol. 2, no. 9, pp. 159-179, 1978.

Yufei Tao obtained the diploma from the South China University of Technology in August 1999 and the PhD degree from the Hong Kong University of Science and Technology in July 2002, both in computer science. After that, he spent a year as a visiting scientist at the Carnegie Mellon University. Currently, he is an assistant professor in the Department of Computer Science, City University of Hong Kong. Dr. Tao is the winner of the Hong Kong Young Scientist Award 2002 from the Hong Kong Institution of Science. His research includes query algorithms and optimization in temporal, spatial, and spatio-temporal databases.

Dimitris Papadias is an associate professor in the Computer Science Department at the Hong Kong University of Science and Technology (HKUST). Before joining HKUST in 1997, he worked and studied at the German National Research Center for Information Technology (GMD), the National Center for Geographic Information and Analysis (NCGIA, Maine), the University of California at San Diego, the Technical University of Vienna, the National Technical University of Athens, Queen's University (Canada), and University of Patras (Greece). He has published extensively and been involved in the program committees of all major database conferences, including SIGMOD, VLDB and ICDE.