Available online at www.sciencedirect.com
ScienceDirect
Information Sciences 177 (2007) 607–631
INFORMATION SCIENCES
AN INTERNATIONAL JOURNAL
www.elsevier.com/locate/ins

Cost modeling of spatial operators using non-parametric regression ☆

Songtao Jiang [a], Byung Suk Lee [a,*], Zhen He [b]

[a] Department of Computer Science, University of Vermont, Room 323 Votey Building, Burlington, VT 05405, USA
[b] Department of Computer Science, La Trobe University, Bundoora ,Vic. 3086, Australia

Received 8 November 2004; received in revised form 23 April 2006; accepted 29 April 2006

Abstract

In an object-relational database management system, a query optimizer requires users to provide cost models of user-defined functions. The traditional approach is analytical, that is, it builds a cost model generated as a result of analyzing the query processing steps. This analytical approach is difficult, however, especially for spatial query operators because of the complexity of the processing steps. In this paper, a new approach that uses *non-parametric regression* is proposed. This approach significantly simplifies the process of building a cost model, while achieving highly accurate cost estimation. We demonstrate the simplicity and efficacy of this approach through experiments for three spatial operators—the range query, the window query, and the *k*-nearest neighbor query—commonly used in spatial databases, using both real and synthetic data sets.

Keywords: Cost model; Non-parametric regression; Spatial database; Spatial operator

1. Introduction

Spatial data are important in many areas of the public and private sectors, including geographical information systems [32,60], computer-aided design [19,42], multimedia information systems [5,13], and earth observing systems [38]. Recently, with the development of new technologies for large-scale data management [52,54], it is becoming increasingly feasible to store and manage a very large volume of spatial data. As a result, it has become increasingly important to efficiently execute queries on spatial data. In this regard, there has been significant research into spatial query optimization [34,49].

In most database management systems (DBMSs), query optimizers use a cost-estimate-based approach to generate an optimal query execution plan. In this approach, the query execution cost is estimated using cost

☆ This work was partially carried out while the author was at the Department of Computer Science, University of Vermont.

* Corresponding author. Tel.: +1 802 656 1919; fax: +1 802 656 0696.
E-mail addresses: songtao.jiang@uvm.edu (S. Jiang), byung.lee@uvm.edu (B.S. Lee), z.he@latrobe.edu.au (Z. He).

0020-0255/$ - see front matter
doi:10.1016/j.ins.2006.04.010

models of individual operators. To this end, there has been significant research into developing cost models of spatial operators, such as spatial selection operators, spatial join operators, and spatial aggregation operators [1–3,7,8,12,23,24,50,51].

Traditionally, cost models are built *analytically* in the form of mathematical functions of various data profile parameters (e.g., cardinality, selectivity) and system parameters (e.g., page size, buffer size). However, as explained in Section 3.1, building cost models for these spatial operators using an analytical approach is a very involved process. Thus, cost models are typically drastically simplified and approximated; this results in inaccurate cost estimations of individual operators and, consequently, a poor query execution plan. Indeed, as pointed out by Gardarin et al. [30] and Lanzelotte et al. [43], the objective of using such an analytical model for a query optimizer is "not to find the best execution plan but rather to avoid the bad ones." Thus, database system administrators frequently turn off the cost-estimate-based query optimizer because of its poor performance and use only the heuristics-based query optimizer instead.

A new method is proposed here that is simple and effective for generating highly precise cost models. A cost model is built using a *statistical* approach. That is, a cost data set is first generated by repeatedly executing a spatial operator, and then a statistical model of the data set is built. This statistical modeling is much simpler than analytical modeling, as it does not require any understanding of the system's internal query-processing mechanism, but needs only the cost data set generated from query executions.

To use this approach, users only have to provide information that is necessary to generate a cost data set. As explained in Section 4, this requires only basic knowledge of a spatial operator, such as the semantics of what the operator retrieves given the values of its input arguments. Thus, the main objective of this paper is to show how simple the statistical cost-modeling approach can be and how precise the cost model generated may be, despite the complexity of spatial query processing inside a DBMS.

In this paper *non-parametric regression* is used as the statistical modeling technique. Regression is adequate here owing to its inherent ability to fit a model to noisy data. It is imperative to use a non-parametric as opposed to a parametric technique, because the *arbitrary* cost-variations typical of spatial operators make it infeasible to provide a parametric model. This issue is revisited in Section 5. In the experiments, thin-plate spline fitting [14,47,65] is used as an example of a non-parametric regression technique.

The experiments used a real DBMS—Oracle Spatial™ —for the spatial query processing, and the following three common spatial search operators: the *range query*, the *window query*[1], and the *k-nearest neighbor query*. The R-tree index was utilized for all three operators. The experimental results, obtained using both synthetic and real spatial data sets, confirm the validity of our argument on the simplicity and efficacy of our statistical modeling technique.

In this approach, each cost model is tied to the spatial data set used to build the model. This is particularly useful in an environment in which there is little or no update of the data set. In this case, the cost of training a cost model is amortized over repeated use of the model. Such an environment is quite common for spatial databases. One example is a cartographic database used in geographic information system applications. Considering an urban map as a specific example, roads and buildings in the map hardly change and, if they do, the changes are infrequent and mostly trivial (e.g., one building, one intersection). In an environment with frequent updates, the robustness of a model becomes an important issue. This is beyond the scope of this paper.

The main contributions in this paper are: (1) to propose a statistical non-parametric regression technique for generating the cost model of a spatial operator and (2) to demonstrate how simple and effective the proposed technique is, despite the complexity of spatial query processing. The applicability of this approach is not limited to the query optimizer. It can also be used for other applications that require execution cost modeling, such as dynamic resource allocation in parallel processing and price negotiations between a mobile agent and a hosting site.

The remainder of the paper is organized as follows. Section 2 provides background information. The existing cost modeling approaches are reviewed in Section 3. Section 4 describes the three commonly used spatial

[1] The terms *range query* and *window query* are often used interchangeably. However, they are distinguished here, as explained in Section 4.

operators. The statistical cost model-building procedure is then outlined in Section 5. The cost-modeling approach is empirically evaluated in Section 6 and Section 7 concludes.

2. Background

This section gives an overview of some relevant topics for spatial database systems and non-parametric regression.

2.1. Spatial database systems

This subsection briefly describes the spatial data (or object) representations and relationships, spatial index structures, and spatial queries and their processing.

2.1.1. Spatial data

Spatial data are more complex in their structures than traditional business data and, thus, are modeled using complex geometry types, complex operations, and complex access methods [29]. The actual complexity varies for different applications, such as geographical information systems (GIS), VLSI CAD, mechanical CAD, and image analysis. Since GIS is a primary motivation for spatial databases [60], we base our discussion on GIS here.

In the Open GIS standard Simple Features Specification for SQL [69], a geometry type has four categories, namely, *point*, *curve*, *surface*, and *geometry collection*. Different geometry types represent spatial objects of different shapes. A *point* is a zero-dimensional object, and is described with the coordinates of a vertex denoting the point. A point models, for example, the location of a city in a map. A *curve* is a one-dimensional object and is described with a line string; a line string is made of one or more line segments that do not form a closed loop. Each line segment connects two points. A line string models, for example, a highway on a map. A *surface* is a two-dimensional object, and is described with a polygon; a polygon is made of one or more line segments that form a closed loop. A polygon models, for example, a stadium on a map. The shape of a polygon may be irregular. A *geometry collection* models a complex shape such as a group of islands, and is described with points, curves, surfaces, and other geometry collections.

There are many possible topological relationships between spatial objects, but the following are most commonly used in GIS [22]: *disjoint*, *meet*, *overlap*, *contains*/*inside*, *covers*/*covered_by*, and *equal*. Fig. 1 illustrates the first four relationships. For the other two relationships, the *covers*/*covered_by* is a special case of contains/inside, whereby a certain part of the contained object meets the containing object; the relationship *equal* is a special case of covers/covered_by, whereby the covered object and the covering object have the same geometrical shape. These topological relationships can be determined using a distance function that computes the geometrical distance between two objects in a spatial data space (typically the Euclidean space).

2.1.2. Spatial indexes

Various types of spatial indexes have been proposed to date [6,9,33,36,45,55,59,63]. Among these, the R-tree [33] is commonly used in spatial DBMSs, and has been used in our work as part of Oracle Spatial™. In the R-tree, a spatial object is represented by its minimum bounding rectangle (MBR). An MBR is a rectangle that tightly encloses one or more spatial objects. For a *two*-dimensional MBR (which is typical of a spatial application),

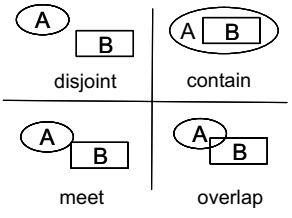

Fig. 1. Examples of topological relationships.

its location is specified with two pairs of coordinates: the coordinates of the lower left corner $\langle x_{\min}, y_{\min} \rangle$ and the coordinates of the upper-right corner $\langle x_{\max}, y_{\max} \rangle$. Spatial objects are then inserted, deleted, and searched based on the locations of their MBRs. Fig. 2 shows an example R-tree index created on spatial objects.

In the case of the Oracle Spatial used in our work, the R-tree is structured using relational tables [40,41]. Such an R-tree structure is called a *relational R-tree* in Ref. [41]. According to Kriegel et al. [41], a relational R-tree comprises three tables: an index table, a user table, and a meta table. Fig. 3 illustrates these tables for the R-tree shown in Fig. 2. The *index table* (Fig. 3a) stores a mapping from the R-tree index to a relational table. It has the following four columns: Page_id (logical node identifier), Page_level (the node height in the tree structure), Child_id (the node identifier of the connected entry), and Child_mbr (the child node's MBR).

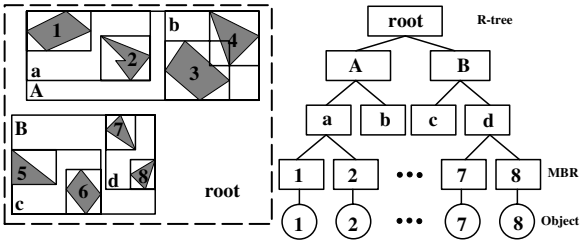

Fig. 2. An example of an R-tree index.

example_idx_table

Page_id	*Page_level*	*Child_id*	*Child_mbr*
Root	4	Root	Rect((0,0),(100,100))
Root	3	A	Rect((10,0),(90,40))
Root	3	B	Rect((5,60),(60,95))
A	2	a	Rect((10,0),(50,30))
A	2	b	...
B	2	c	...
B	2	d	...
a	1	MBR1	...
a	1	MBR2	...
...	...	...	...
MBR1	0	Object1	Rect((10,0),(30,20))
MBR1	0	Object2	...
...	...	...	...

(a)

example_usr_table

ObjectID	Geometry
Object1	polygon((15,0),(30,10),(17,20),(10,10))
Object2	polygon((32,8),(50,10),(48,30),(35,20),(40,22))
Object3	polygon((58,15),(80,30),(68,40),(55,33))
---	---

(b)

Index_Name	User_table	Index_table
'example_idx'	'example_usr_table'	'example_idx_table'
---	---	---

(c)

Fig. 3. An example of the structure of a relational R-tree index. (a) Index table, (b) user table and (c) meta table. (The origin of the coordinate for Rect is in the upper-left corner.)

In the figure, MBRs 1–8 are the leaf nodes of the R-tree. These MBRs are at Page_level 0, and their parent MBRs are at Page_level 1, and so on. The column Child_id contains the Page_id for each entry, while the column Child_mbr stores the MBR information. Thus, the columns Page_id and Child_id together comprise the primary key of the table. By searching the index table recursively for a certain area, the corresponding spatial objects (i.e., Child_id values of the leaf nodes) are found. Then the actual spatial objects are retrieved from the user table. The *user table* (Fig. 3b) stores geometric information about the actual spatial objects indexed—for example, the coordinates of the polygons representing spatial objects. The *meta table* (Fig. 3c) stores information about the indexes and the tables representing them. That is, it contains information about the user and index tables of an R-tree index.

2.1.3. Spatial queries

Since spatial objects have topological relationships between them, users can use spatial operators to retrieve objects that satisfy a certain topological relationship. (See Section 2.1.1 for commonly used topological relationships.) We classify spatial operators into the following three categories based on the classifications by Gaede and Günther [29] (and in the book by Shekhar and Chawla [60]): *selection*, *join*, and *aggregation*. Spatial selection involves finding all objects satisfying a certain selection condition, and can be further classified into *point-based* selection and *region-based* selection. The former identifies all objects containing a given point; the latter identifies all objects satisfying a specified topological relationship with a given object (or, precisely, the region enclosing a given object). A spatial join between two given sets of objects identifies all pairs of objects—one from each set—satisfying a specified topological relationship between the sets. A spatial aggregation identifies the objects that satisfy a certain condition on the aggregation of objects that have a certain topological relationship with a given object.

There are several such spatial operators commonly supported by spatial DBMSs. These include range queries, window queries, and k-nearest neighbor queries. A range query retrieves objects within a given range from a given reference point or object. A window query retrieves objects contained in or overlapping a given window. These two queries are considered to be spatial selection operators [60]. A k-nearest neighbor query retrieves the k objects nearest to a reference point. This query is considered to be a spatial aggregation operator [60]. (Details of these operators are given in Section 4.)

Executing these spatial operators involves computing and comparing the geometrical distances from one point or object to other objects. Spatial DBMSs use a *two-tier query model* [60] to do this efficiently. As shown in Fig. 4, the two-tier query model has two steps: the *filter* step and the *refinement* step. The filter step uses the spatial index to select possible candidate objects that may be in the query results. Then the refinement step performs exact comparisons among the candidate objects to select those that should be in the final query result. The filter step involves only an index search and is considered a lower-cost step. In contrast, the refinement step is computationally more expensive, but the computations are applied only to a smaller set of objects returned from the filter step.

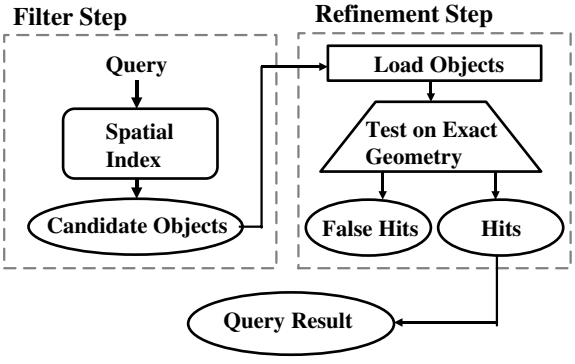

Fig. 4. Two-step query processing (source: Brinkhoff et al. [15]).

Query processing in terms of the MBRs in the R-tree index for the range query is reiterated as an example. In the filter step, the MBRs overlapping the MBR of the circle representing the query range are selected. Because the MBRs are rectangles, it is computationally less expensive than checking if the spatial objects (e.g., irregular polygons) are in the query range. In the refinement step, the objects enclosed in the selected MBRs are then retrieved from disk and stored in memory. The exact geometry of each object, and not the MBR, is then used to check if the object indeed overlaps the circle. This step usually requires a CPU-intensive geometric computational algorithm.

2.2. Non-parametric regression

This subsection provides a brief overview of non-parametric regression with a focus on thin-plate spline-based fitting, which is supported by the SAS/STAT package used in our work. Since the focus here is not on the theoretical principles of non-parametric regression, only a limited discussion is provided in this subsection. Interested readers may obtain more in-depth information from the literature [14,20,21,47,64,65].

2.2.1. Basic concepts

Consider a training data set of n data points $\{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_n, y_n)\}$ where, for each data point i ($i = 1, 2, \ldots, n$), $\mathbf{x}_i$ ($\equiv(x_{i1}, x_{i2}, \ldots, x_{ip})$) is a vector of the values of p independent variables (or predictors) and y_i is the value of the corresponding dependent variable (or response). Regression then builds a model by fitting the data points to a regression function. In the case of parametric regression, the fitting involves computing the parameters of a regression function f of the following form:

$$y_i = f(\boldsymbol{\alpha}^{\mathrm{T}}, \mathbf{x}_i) + \varepsilon_i \quad \text{for } i = 1, 2, \ldots, n, \tag{1}$$

where $\boldsymbol{\alpha}^{\mathrm{T}}$ is a transposed vector of the parameters (i.e., $\alpha_1, \alpha_2, \ldots, \alpha_p$) to be computed and ε_i is the random error of data point i. The distributions of ε_i, $i = 1, 2, \ldots, n$, are each Gaussian with zero mean and are independent of one another. The function f relates the average of the dependent variable values ($y_1, y_2, \ldots, y_n$) to the predictors ($\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n$).

In contrast, the function required for non-parametric regression is built as a result of the fitting. That is, given a target regression form

$$y_i = f(\mathbf{x_i}) + \varepsilon_i \quad \text{for } i = 1, \ldots, n \tag{2}$$

the regression function f is estimated directly. Non-parametric regression seeks to trace the average value of y_i as a function of $\mathbf{x}_i$ with minimum assumptions about the form of the relationship.

Various approaches can be used for building a non-parametric regression model, such as local regression [20,21], regression trees [31], spline-based fitting [53,64], kernel smoothers [56,66], kNN regression [46,61], orthogonal series estimators [16,67], and projection pursuit [35]. These all allow for great flexibility in the possible form of the regression surface and make no assumption about the parametric form of the model.

The current SAS package supports two non-parametric regression techniques: *thin-plate spline-based fitting* [14,47,65] (called TPSPLINE) and *local regression* [20,21] (called LOESS). In the remainder of this section, we give a brief overview of these two techniques, adapted from Ref. [70]; the rationale for the choice of TPSPLINE is then explained.

2.2.2. Thin-plate spline-based fitting

Spline-based fitting has been used as a non-parametric regression analysis tool since 1998 [64]. TPSPLINE uses a type of spline called a thin-plate spline; this name refers to the physical analogy of bending a thin sheet of metal. This technique uses the penalized least-squares method to estimate multivariate regression surface using thin-plate smoothing splines. Using the notation of the penalized least-squares estimate, the function f can be obtained by minimizing the following quantity:

$$\frac{1}{n}\sum_{i=1}^{n}(y_i - f(\mathbf{x}_i))^2 + \lambda J_{m,p}(f) \tag{3}$$

where the first term measures the *goodness of fit*, and the second term measures the *smoothness of f*.

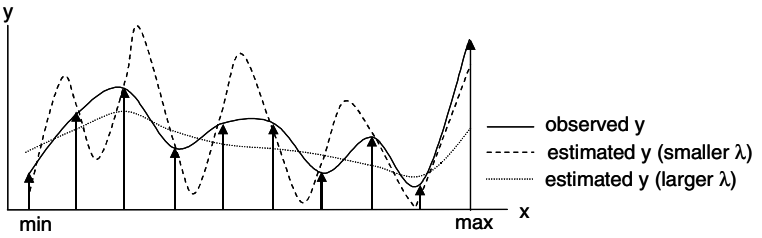

Fig. 5. Goodness of fit and smoothness for different values of the smoothing parameter (λ).

The multiplier λ in Eq. (3) is a *smoothing parameter*, which regulates the weight between the smoothness and goodness of fit. There is a tradeoff between these two measures, depending on the value of λ. Specifically, a smaller value of λ results in better fitting of the training data points, whereas a larger value results in smoother fitting across the model space (i.e., the effective parameter space). Fig. 5 illustrates this tradeoff, where the vertical arrows are training data points, which are samples of the observed (i.e., actual) values shown as a solid line. The broken line shows the estimated values for a smaller λ, whereas the dotted line shows the values for larger λ. As λ becomes smaller, the fitting becomes more accurate (i.e., "better") at the training points, but less accurate at other points (used as testing points) because the fitting fluctuates more. The converse is true as λ becomes larger, because the fitting becomes smoother.

The optimal value of λ can be chosen by minimizing the function $V(\lambda)$, called a generalized cross-validation (GCV) function [64]

$$V(\lambda) = \frac{\left(\frac{1}{n}\right)\|(\mathbf{I} - \mathbf{A}(\lambda))\mathbf{y}\|^2}{\left[\left(\frac{1}{n}\right)\mathrm{tr}(\mathbf{I} - \mathbf{A}(\lambda))\right]^2}, \tag{4}$$

where $\mathbf{A}(\lambda)$ is called the smoothing matrix with the fit $\mathbf{y}_\lambda = \mathbf{A}(\lambda)\mathbf{y}$.

$J_{m,p}(f)$ in Eq. (3) is the *penalty on the roughness* of f, and is defined as an integral of the square of the mth derivative ($m \geqslant 2$) of f as

$$J_{m,p}(f) = \int_{-\infty}^{\infty} \cdots \int_{-\infty}^{\infty} \sum \frac{m!}{\alpha_1! \cdots \alpha_p!} \left[\frac{\partial^m f}{\partial x_1^{\alpha_1} \cdots \partial x_p^{\alpha_p}}\right]^2 \mathrm{d}x_1 \cdots \mathrm{d}x_p, \tag{5}$$

where $\alpha_1 + \cdots + \alpha_p = m$.

The estimate of f, denoted as $\hat{f}$, is obtained as a linear combination of a sequence of basis functions, that is

$$\hat{f}(\mathbf{x}_i) = \theta_0 + \sum_{j=1}^{p} \theta_j x_{ij} + \sum_{j=1}^{n} \delta_j B_j(\mathbf{x}_i), \tag{6}$$

where B_j is a basis function that depends on $\mathbf{x}_j$, and θ_j and δ_j are coefficients to be estimated.

Readers interested in the algebraic crux of the thin-plate spline-based fitting algorithm are referred to p. 570 of an article by Brookstein [14].

2.2.3. Local regression

In LOESS, the regression function f (see Eq. (2)) can be locally approximated as a polynomial function [20,21]. For univariate LOESS, the regression fits at a selected local point x_0 as

$$y_i = \beta_0 + \beta_1(x_i - x_0) + \beta_2(x_i - x_0)^2 + \cdots + \beta_p(x_i - x_0)^p + \varepsilon_i \tag{7}$$

while using weighted least squares with kernel weights $K(b^{-1}(x_i - x_0))$ to minimize [58]:

$$\sum_{i=1}^{n} \{y_i - \beta_0 - \cdots - \beta_p(x_i - x_0)^p\}^2 K\left(\frac{x_i - x_0}{b}\right). \tag{8}$$

Specifically, data points are fitted using weighted least squares at the centers of neighborhoods. For each neighborhood, it contains a specified percentage of the data points. This fraction of the data is called the

smoothing parameter, which controls the smoothness of the estimated surface. The number of points used to estimate the regression function $f(x)$ is fixed, regardless of the estimation location of x_0.

2.2.4. Choice of TPSPLINE

While the preliminary study showed that both TPSPLINE and LOESS are suitable, TPSPLINE was chosen over LOESS. The reason for this is that TPSPLINE is mathematically more elegant, gives slightly better surface smoothing, and can handle larger data sets. For instance, since TPSPLINE uses the penalized least-squares method to fit the data with a flexible model, the number of effective parameters (i.e., the number of sample data points used by TPSPLINE as the unique design points) can be as large as the number of unique design points. Thus, as the sample size increases, the model space increases as well, thus enabling TPSPLINE to fit more complicated situations [70]. In addition, TPSPLINE can fit both semi-parametric and non-parametric models, which provides great flexibility in building models. (Only a non-parametric model was used because a test showed that the fitting error was higher when a semi-parametric model was fitted using TPSPLINE.) On the other hand, LOESS is considered more suitable than TPSPLINE if there are many outliers in the data and the fitting has to be robust [25], and is less expensive to compute [58].

3. Existing cost-modeling approaches

Existing cost-modeling approaches can be categorized as analytical or statistical approaches. An analytical approach involves a highly complicated analysis of cost-incurring operations, and this requires the user to have advanced knowledge of the database system—internal implementation of the DBMS, system configurations, internal mechanisms of functions and operators, etc. In reality, however, most users do not have this type of knowledge. In this regard, a statistical approach is promising because of its simplicity. , as it only requires the user to "help" the system to collect a statistical training data set.

3.1. Analytical cost modeling

Owing to the inherent complexity of analytical cost modeling, simplifications and approximations are unavoidable in the analysis of most existing studies. This makes the resulting cost model imprecise, and thus applicable to only limited or unrealistic cases of spatial data sets. This subsection points out the simplifications/approximations made in some previous studies and discusses them.

Acharya et al. [2], Belussi and Faloutsos [7], and Faloutsos and Kamel [23], proposed a cost model for the window query, where both query windows and spatial objects are *rectangles*. In this case, there is no significant computation needed in the refinement step, since it only involves comparing rectangles. Hence, only the cost of the filter step was considered and the refinement step was ignored. Aboulnaga and Naughton [1] proposed a more general cost model for the window query, where both query windows and spatial objects are *general polygons*. In this case, a significant CPU cost is incurred in the refinement step and, therefore, the cost model considers both filter and refinement steps. In the filter step, the access method is either a sequential scan or an R-tree search. Hence, the cost of this step depends on the access method used. The authors assume that the cost of the refinement step is independent of the access method used for the filtering. This assumption ignores the intricacy of internal processing such as pipeline operations. Furthermore, the authors use the MBR of a general polygon as the query window in the filter step, thus in effect approximating the filtering cost to that of a rectangular window query. Note that in our experiments we also use general polygon-shaped spatial objects (see Section 6.3.1), therefore incurring significant cost in the refinement step, but the statistical cost modeling reflects both filter and refinement steps *as an integral* operation without involving the analysis required for analytical cost modeling.

Typically, an index structure (e.g., an R-tree) is used to speed up the filter step. In this regard, there has been extensive research on analysis of an index structure to model the cost of the filter step. Some previous studies are discussed in the remainder of this subsection to show the complexity of such an analysis and the simplifications made.

3.1.1. Analytical cost modeling of the filter step

Friedman et al. [27] proposed a cost model for the nearest neighbor query using the KD-tree. The analysis is based on the maximum metric—a metric that bisects the data space along the dimension with the longest range. In general, this metric allows for only an approximate analysis. Moreover, the resulting model estimates only the number of leaf node accesses, which does not closely reflect the actual cost.

Faloutsos et al. [24] carried out a cost analysis for window queries (including range queries) using the R-tree and the R+-tree. This paper provides the first known analysis of the R-tree, and compares the R-tree and R+-tree analytically. The resulting cost models estimate only the number of data page accesses. As an example, the analysis needed to model the expected number of data page accesses in the R-tree is sketched, as presented by Faloutsos et al. [24].

Example 1. Suppose the database stores N spatial objects in a d-dimensional data space, and each data page stores an average of C objects. Assume a query accesses a hypercube H_D of volume $V_D = 1/N$ in the data space, and approximates the page region mapped from the hypercube H_D as a hypercube H_P with volume $V_P = C/N$. Then, the page region is accessed if and only if H_P intersects with H_D. The probability of their intersecting in one dimension can be approximated as $\sqrt[d]{V_D} + \sqrt[d]{V_P}$ for very large N. For simplicity, assume the dimensions of the data space are not correlated. Then, for very large N, the probability in the d-dimensional data space is $\left(\sqrt[d]{V_D} + \sqrt[d]{V_P}\right)^d$, and hence the expected number of data page accesses is the multiplication of this probability and the number of data pages, N/C, in the database, and hence $(\sqrt[d]{V_D} + \sqrt[d]{V_P})^d N/C$.

Despite the simplifying assumptions and approximations made, this analysis is quite complicated for most users. Furthermore, the resulting cost model has several limitations. First, the model is not applicable to a sparse database because the model assumes that the number of objects is very large. Second, the effects of the correlations between dimensions are not considered. Third, the boundary effects are not reflected in the model. (Here, a boundary effect refers to the change in cost as the query reaches the boundary of the data space.) Indeed, significant impacts on the costs are reported for the correlation effects by Faloutsos and Kamel [23] and for the boundary effects by Pagel et al. [50] and Berchtold et al. [8]. Fourth, a page region is approximated as a hypercube, which is not necessarily valid.

Berchtold et al. [8] proposed cost models geared for a high-dimensional data space and considering the boundary effects (but not the correlation effects) for the range query and the nearest neighbor query. The boundary effects are more significant in a higher-dimensional space. Their models, however, assume that the number of data pages is a power of two and that the data pages cover the entire data space completely without any overlap. These assumptions are not realistic at all. The number of data pages can be any number, and an empty region of the data space has no data page to store its non-existent objects. Böhm [12] extended the models proposed by Berchtold et al. [8] to consider the correlation effects as well (using the concept of the fractal dimension [23]), and added a cost model for the k-nearest neighbor query. These models, however, assume a smooth spatial data distribution. This is not a valid assumption in general, as shown in our experiments (see Fig. 10 in Section 6.3.1, for example).

3.1.2. Precision of an analytical cost model

To the best of our knowledge, most research on analytical cost modeling reports *absolute* cost values (e.g., the number of disk page accesses) instead of error statistics. An example is the analytical cost model of a nearest-neighbor (i.e., kNN with $k = 1$) query used by Korn et al. [39] in 2001. It shows the cost in terms of the number of leaf accesses in the R-tree as follows: for three different data sets, the predicted (i.e., estimated) costs are 5.18, 3.05, and 19.63, respectively, and the mean/standard deviation of the observed costs are 6.7/6.08, 9.82/9.88, and 32.8/44.17, respectively. This amounts to an error in the range of 29–222%, even if we consider only the average costs as a best-case representation of their results.

3.2. Statistical cost modeling

Recently, statistical approaches have been studied in building the cost model of a user-defined function (UDF) for a DBMS query optimizer. These approaches have several advantages over the analytical approach. First, the cost modeling does not require users to have advanced knowledge of the database systems; users

only need to determine from the input arguments of the UDF the variables that significantly influence the cost (called *model variables*) and to provide a simple specification for collecting training data needed to build a cost model. Second, the cost modeling for the UDF is carried out as one logical step instead of its individual steps and is therefore less involved. Third, the cost modeling considers all possible cost-incurring factors naturally through statistical fitting, thereby generating a highly precise model.

Boulos and Ono [11] built a multi-dimensional histogram based on a training data set collected through UDF executions. The histogram constructed is then stored in the DBMS and used as the cost model. The UDFs used in their experiments are text search functions, e.g., Contains(text, keyword_list) for retrieving documents containing the keywords in keyword_list from the documents in text. In this approach, however, the data set becomes too large to be stored and processed with reasonable performance. The authors mitigate this problem by sampling the data set generated; however, this results in high errors (55–79% relative error) in the cost estimation. Compared with this approach, the model data generated by our approach is very small—only the coefficients of the regression model. This confers an advantage, particularly in a query optimization environment, because a query optimizer should operate with a limited amount (as small as a few kilobytes) of memory.

Lee et al. [44] used parametric regression as the modeling technique. The UDFs are aggregate functions of financial time series, such as Median, MinMavg (minimum moving average), NthMavg (*n*th moving average). For these UDFs, a quadratic model was suitable for achieving accurate cost estimations (less than 5% median relative error and 10% mean relative error in the majority of cases). This approach, however, requires the user to provide a parametric model and only tunes the coefficients. Thus, it is not generally applicable to other UDFs with more complex cost variations, such as spatial operators. In contrast, the non-parametric regression used in our approach can fit cost variations of any shape without the user's involvement, and is thus generally applicable to various UDFs.

4. Three common spatial operators

This section describes three common spatial operators: the range query, the window query, and the *k*-nearest neighbor query. The focus here is on showing the complexity of analyzing the processing costs.

Fig. 6 illustrates the three spatial operators on the objects shown in Fig. 2. The *range query* finds objects contained within or overlapping a circle with a given reference point as the center and a given distance as the radius. To use this operator, two arguments need to be provided: the reference point and the distance. (If the reference is given as an object instead of a point, then the center of its MBR, or a certain vertex of the object, may be used as the reference point.) The *window query* finds all objects that intersect a given rectangular window. Two points need to be provided to specify the window. In a two-dimensional space, they can be the points at the bottom-left corner and the top-right corner. The range query may be considered as a window query with a circular window. However, they are distinguished here because the cost models have a different number of model variables (as described in Section 6.1). The *k-nearest neighbor query* finds the *k* nearest neighbors of a given reference point in the Euclidean space. Thus, two arguments need to be provided: the reference point and the number *k*.

An analytical approach to building cost models for these three spatial operators requires analysis of the performance at a given page granularity for the disk IO cost and in seconds (or finer, e.g., milliseconds) for the CPU cost. This analysis is carried out as described in Section 3.1.

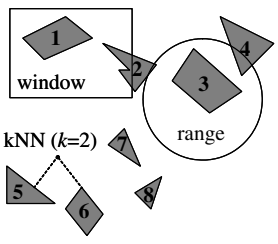

Fig. 6. Three common spatial operators.

According to the two-step query-processing strategy (described in Section 2.1.3), the costs of these spatial operators depend on many factors. First, the distribution of spatial objects affects the index tree structure (e.g., tree height, node fanout) built on them, which directly influences the index search performance (e.g., the number of MBRs accessed during the search). Second, the input argument values of a spatial operator affect the index search performance and the number of candidates selected in the filter step. Third, the index search algorithm associated with the index tree structure has a fundamental effect on the index search performance. For example, in the case of the *k*-nearest neighbor query, the index search performance varies depending on which of the different search algorithms [4,17,26,28,37,57,68] is used on which of the different spatial index structures [6,9,33,36,45,55,59,63]. All these factors combined significantly influence the index tree search cost, MBR comparison costs, and the number of candidates selected in the filter step, which in turn affects the cost of the refinement step.

Moreover, exact geometrical comparison in the refinement step is an involved process and, thus, complicates the cost analysis further. The comparisons involve arbitrarily shaped objects, the boundaries of which are represented by points or line strings. The complexity of these geometrical shapes has a direct impact on the exact comparison cost. For example, in the case of the range query, to examine the topological relationship between the reference point and a spatial object, *each line segment* in the geometrical boundary of the spatial object needs to be examined to find if the spatial object is within the query distance from the reference object (or point). Note that the costs are also affected by system configurations such as the buffer size and buffer management policy for both the filter step and the refinement step.

Analytical cost modeling considering all these factors is very difficult, if not impossible. Moreover, as described in Section 2.1.2, the spatial index R-tree is structured as relational tables in Oracle Spatial™. Hence, the spatial index search uses relational join operations, which themselves may utilize indexes on columns. Thus, the analysis would be even more involved than analyzing the traditional tree-structured spatial index.

5. Procedure for building a statistical cost model

In this section, the procedure for building a statistical cost model for spatial operators is described. Note that the novelty is the statistical approach as a whole applied to the problem of spatial operator cost modeling, instead of the individual steps in the procedure.

The approach involves four steps, as shown in Fig. 7. The first step is to determine the model variables for a spatial operator. The second step is to prepare a set of the sample values of model variables, which is used in the third step to generate a model training data set. This in turn is used in the fourth step to train the statistical cost model. (Fig. 11 in Section 6 shows an example of a model training data set and the surface fitted using a trained cost model.) Each step is now described in more detail.

5.1. Determining model variables

Model variables determined in this first step directly affect subsequent steps in the cost model building procedure. As mentioned in Section 3.2, model variables significantly influence the cost of a query operator in a database system. There may be many factors that influence the cost to a different degree, but our educated assumption is that only a small number (i.e., 1–5) of them have a dominant influence. Identifying the right model variables is very important, as the model variables can influence the time required to generate the training data set, the model building time, and, more importantly, the model precision. Users can determine model variables from their understanding of the spatial operator. Step 1 in Fig. 8 shows example model variables (x, y, and d) for the range query. (In Section 6.1 we discuss in detail how model variables are determined for the range query.)

5.2. Preparing a model variable value set

A model variable value set consists of selected values of model variables; the values are sampled from a data space formed by the model variables. In this work, *grid sampling* is used to cover the model space (i.e., the Cartesian space defined by model variables) evenly. (A more sophisticated sampling technique, such as

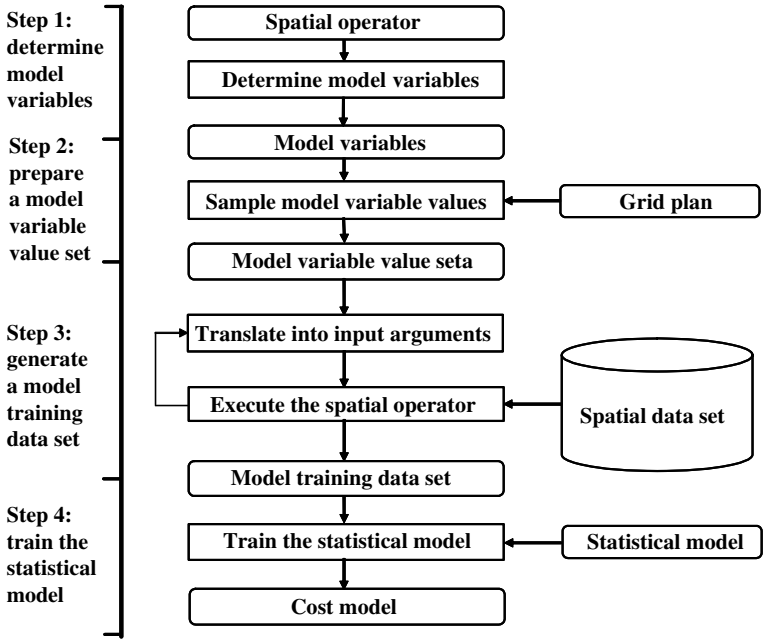

Fig. 7. The steps in the statistical cost model building procedure.

Spatial operator: Range (p, d)

Input arguments: p, d

- p: reference point
- d: distance from the reference point

Step 1: determining model variables

Output: model variables x, y, d

- x: x-coordinate of the reference point
- y: y-coordinate of the reference point
- d: distance from the reference point

Step 2: preparing a model variable value set

Output: $\{<v_x, v_y, v_d>\}$

- v_x: a sample value of the model variable x
- v_y: a sample value of the model variable y
- v_d: a sample value of the model variable d

Step 3: generating a model training data set

Output: $\{<v_x, v_y, v_d, cpu, io>\}$

- v_x: a sample value of the model variable x
- v_y: a sample value of the model variable y
- v_d: a sample value of the model variable d
- cpu: CPU cost of executing Range(Point(v_x, v_y), v_d) where Point(v_x, v_y) is a reference point at the coordinates $<v_x, v_y>$.
- io: disk IO cost executing Range(Point(v_x, v_y), v_d) where Point(v_x, v_y) is a reference point at the coordinates $<v_x, v_y>$.

Fig. 8. An example of generation of a training data set.

density-biased sampling [48] would be more effective for achieving higher accuracy. The particulars of the sampling techniques are, however, beyond the scope of this paper.) The grid is determined according to a *grid plan*, which is a specification for partitioning the model space. Different spatial operators may need different grid plans. The grid plan used in experiments is described in Section 6.3.2. Step 2 in Fig. 8 shows an example

set of model variable values for the three model variables of the range query. Each combination of v_x, v_y, and v_d constitutes a grid point.

5.3. Generating a model training data set

A model training data set is generated by repeatedly executing a spatial operator according to a grid plan. The input argument values of the spatial operator are translated from the model variable values (generated in the previous step). Each input argument of the spatial operator is either a model variable itself or derived from one or more model variables. Step 3 in Fig. 8 shows an example of generation of a training data set from the model variable value set generated in the previous step. To generate input arguments of the range query, the model variables x and y are translated into the input argument p (reference point) as $Point(x, y)$, and the model variable d becomes the input argument d (distance). The training data set thus generated is formed by combining the model variable values (e.g., v_x, v_y, v_d) and the measured CPU and disk IO costs (cpu, io), that is, as $\{\langle v_x, v_y, v_d, cpu, io\rangle\}$.

5.4. Training a statistical model

A training data set is used to build a cost model through statistical training. As mentioned in Section 1, non-parametric regression is used as the statistical modeling technique, which fits the training data set with high precision, has a high data/model compression ratio, and is fast at predicting the dependent variable value (i.e., estimating the cost). Parametric regression is not suitable because it requires users to provide a parametric model, which is not possible without knowing the shape of cost variations in the model space. In the case of spatial operators, the shape can be quite complex and arbitrary because the costs are affected significantly by the distribution of spatial data objects (as described in Section 4), which itself is quite complex and arbitrary.

As mentioned in Section 2.2, the SAS TPSPLINE procedure was used for non-parametric regression. Thus, a cost model can be built by simply providing a training data set as an input to the procedure. Fig. 9(a) shows an example training data set (named "train_r_1x4_1") for the range query; the INPUT statement specifies the five columns (three model variables, x, y, and d, and two costs, CPU time and disk IO) of records in the DATALINES statement.

Fig. 9(b) shows an example of using the TPSPLINE procedure to build a model. CPU cost (cpu) and disk IO cost (io) are built separately as dependent variables of x, y, and d. The LOGNLAMBDA option requests the values of the smoothing parameter λ, which is used to compute the GCV values (see Eq. (4) in Section 2.2.2); the data are given as $\log_{10} n\lambda$ values ranging from -4 to -2 in increments of 0.2, where n is the number of training data points. TPSPLINE computes the GCV values for the given list of λ values and picks the one that returns the minimum GCV. The OUTPUT statement specifies that predicted (i.e., estimated) values

```
DATA  train_r_1x4_1
INPUT x y d cpu io;
DATALINES;
    0      0   5000   18    6
 5000      0   5000   25   38
10000      0   5000   24   65
. . .
RUN;
```

(a)

```
PROC TPSPLINE DATA = train_r_1x4_1;
   MODEL cpu io = (x y d)
            / LOGNLAMBDA = -4 TO -2 BY 0.2;
   OUTPUT OUT = estimate PRED LCLM UCLM;
RUN;
```

(b)

Obs	x	y	d	cpu	P_cpu	LCLM_cpu	UCLM_cpu	io	P_io	LCLM_io	UCLM_io
1	0	0	5000	18.00	18.25	18.13	18.37	6.00	6.73	6.53	6.93
2	5000	0	5000	25.00	25.34	25.15	25.53	38.00	40.23	40.01	41.45
3	10000	0	5000	24.00	24.63	24.42	24.84	65.00	63.63	62.55	64.71

. . .

(c)

Fig. 9. Example of statistical model training using the TPSPLINE procedure. (a) A training input data set, (b) training a statistical model and (c) a training output data set.

(PRED) and 95% confidence limits (LCLM for the lower limit and UCLM for the upper limit) are to be saved in an output data set named "estimate" [70]. To make predictions, the values of model variables are used as the input to the trained model. Fig. 9(c) shows an example output; Obs is the observation point number.

6. Evaluations

In this section, the statistical cost-modeling approach is evaluated. First, cost models of the three spatial operators are built according to the steps described in Section 5. Then experiments are conducted to assess the cost estimation accuracy under different conditions. The conditions tested involve different spatial operators, different densities (i.e., the number of objects in a unit space) of the spatial data set, and different sizes of the model training data set.

Specifically, in Section 6.1 model variables are determined for each of the three spatial operators; in Section 6.2 the performance factors considered in designing the experiments are discussed. The set-up of the experiments is described in Section 6.3, including the spatial data sets and the grid plans used to generate model training data sets. Section 6.4 presents the results of the experiments.

6.1. Model variables for the three spatial operators

In this subsection, the model variables for the three spatial operators are determined. Note that in the cost-modeling approach used, it is sufficient to consider only the basic concepts of the spatial operators while ignoring the details of their processing (e.g., index search) internally within the DBMS.

6.1.1. Range queries

Since the range query identifies all spatial objects within a certain distance from the reference point, naturally the cost is influenced by the size and location of the circle defined by the distance and the coordinates of the reference point. Intuitively, given a reference point at $\langle x, y\rangle$, the cost increases as the distance d increases, and given a distance d, the cost varies with the coordinates x and y of the reference point according to the distribution of spatial objects. Thus, the distance d and the coordinates x and y are chosen as the model variables.

6.1.2. Window queries

Like the range query, the cost of the window query is influenced by the size and location of the window. The window in this case is a rectangle defined by the coordinates of the two extreme corners (e.g., lower-left and upper-right) or, equivalently, the coordinates of one corner and the width and height of the rectangle. Intuitively, given the coordinates of the corner at $\langle x_l, y_l\rangle$ (or $\langle x_r, y_r\rangle$), the cost increases as the window size $(x_r - x_l) \times (y_r - y_l)$ increases, and given a window size, the cost varies with the coordinates of the corner (x_l and y_l, or x_r and y_r) according to the distribution of spatial objects. We choose the coordinates x_l and y_l of the lower-left corner and the coordinates x_r and y_r of the upper-right corner as the model variables.

6.1.3. k-Nearest neighbor queries

Since the k-nearest neighbor (kNN) query identifies the k nearest objects from the reference point, the cost is influenced by the size and location of the circle determined by the coordinates of the reference point and the number k. Intuitively, given a reference point at $\langle x, y\rangle$, the cost increases as k increases, and given k, the cost varies with the coordinates x and y of the reference point according to the distribution of spatial objects. Thus, the coordinates x and y and the number k are chosen as the model variables.

6.2. Performance factors

The cost estimation accuracy is used as the primary performance factor for judging the feasibility of this approach. In addition, the cost estimation speed, the model building speed, and the generation speed and size of the training data set are considered as secondary factors. Trade-offs exist among these factors and, thus, we attempt to find a balance among them in the experiments.

6.2.1. Cost estimation accuracy

The *relative error*, calculated as $|actual\ cost - estimated\ cost| \div actual\ cost$, is used as a metric of the accuracy. This accuracy is affected by a number of factors, including the size (i.e., the number of data points) of the model training data set, the dimensionality of the model space (i.e., the number of model variables), and the density of objects in the spatial data set. Specifically, the accuracy is higher if the modeling is carried out using a larger training data set, a lower-dimensional model space, and/or against a lower-density spatial data set. We believe the first two conditions are evident to readers, but the third one may not be. The reason for the third condition is that, if spatial objects are less densely populated, then the actual costs (both CPU and disk IO) are lower because a spatial query needs to process fewer objects; the relative error tends to be amplified if the actual cost (i.e., the denominator) is lower because the absolute residual error (i.e., the numerator, $|actual\ cost - estimated\ cost|$) is not affected by the actual cost.

6.2.2. Cost estimation speed

Since a cost estimation predicts the cost of using a model that has already been built, it is independent of the other performance factors. Unlike model building or the generation of a model training data, cost estimation should be very fast because it is only one of many steps in query optimization, which itself should be very fast.

6.2.3. Model building speed

Model building is carried out off-line and, hence, the speed is not a critical issue. Nonetheless, it should not be excessively long. In experiments, the model building speed is determined by the speed of the SAS procedure. Two factors influence the model building speed: the size of the training data set and the dimensionality of the model space. Since the latter is fixed once the model variables are determined, the model building speed can only be controlled using the former. Naturally, a larger training data set leads to slower model building but a more precise model. This tradeoff is balanced by carefully designing a grid plan for controlling the size of the training data set.

6.2.4. Generation speed for model training data set and size of the data set generated

Like model building, model training is carried out off-line and hence is not under a tight time constraint. However, the generation of a training data set is the most time-consuming of all the steps in Fig. 7, and the generation time increases linearly with the size of the data set generated. Thus, for practicality, this time was limited to 1 h after some trial runs. Evidently, the time limit should vary depending on the cost estimation accuracy required, the size of the spatial data set, the computer system and algorithm speed, etc. As in the case of the model building speed, the tradeoff between the cost estimation accuracy and the generation speed for the training data set and its size is balanced using a grid plan.

6.3. Experimental set-up

6.3.1. Spatial data sets

Four synthetic data sets and one real data set are used as the spatial data in experiments. Each data set is a collection of simple geometric objects distributed in a two-dimensional space.

6.3.1.1. Synthetic data sets. Each synthetic data set covers a two-dimensional spatial region of 10,000 m × 10,000 m, and contains either 1000 or 10,000 objects. The object types are a point, a line string, and a polygon; a point has only one vertex, and a line string and a polygon have three to twelve vertices. The resulting object size is approximately 80 bytes on average and, thus, approximately 100 objects are stored in one disk page (8 KB). The distribution of a data set is either uniform or multi-Gaussian. Each multi-Gaussian data set contains three clusters of objects. Each cluster has the same number of objects distributed with the centroid $\langle m_x, m_y \rangle$ and standard deviation σ set to $(m_x, m_y, \sigma) = (2000, 2000, 2000)$, $(7000, 2000, 600)$, and $(8000, 8000, 1000)$, respectively. Four data sets were generated from the combination of the two object counts and the two distributions:

- DS_1: 1000 objects with uniform distribution
- DS_2: 1000 objects with multi-Gaussian distribution

- DS_3: 10,000 objects with uniform distribution
- DS_4: 10,000 objects with multi-Gaussian distribution

6.3.1.2. Real data set. An urban area map of 66 counties in Pennsylvania [10] was used as the real data set (DS_5). Fig. 10 shows the map of Adams County, one of the 66 counties, as an example. The map projection name is Lambert Conformal Conic, and the resolution is 0.99537 for both the abscissa and the ordinate. The data set contains 15,000 objects in a spatial region of 500,000 m × 300,000 m. The object types are a line string and a polygon, each of which represents a real geometric object. The number of vertices in each object varies widely, from a few to several hundreds, as does the size of each object. The average size of an object is approximately 165 bytes, and hence an average of 50 objects are stored in one disk page (8 KB). Since this is a real data set, the distribution of objects is arbitrary.

6.3.2. Training data sets: grid plans

As mentioned in Section 6.2, a grid plan is used to control the size and generation time of a model training data set. The sampling interval, and hence the grid resolution, determines the number of training points in the model space and, consequently, the size of the training data set.

Two factors are considered when designing a grid plan: the density of a spatial data set and the dimensionality of the model space. First, as explained in Section 6.2, a higher density of objects in a spatial data set leads to higher costs for spatial operators. This in turn incurs a longer time for generating the training data set. If this time exceeds the limit (i.e., 1 h), then the number of training data points should be reduced by lowering the grid resolution, which compromises the cost estimation accuracy as a result. On the other hand, a lower density of objects in a spatial data set allows for higher grid resolution (which results in a larger number of training data points and, consequently, more accurate cost estimations) as long as the generation time for the training data set is within the limit. Second, a higher-dimensional model space lowers the density of the model training data set, since data points in the space become increasingly sparse as the dimensionality increases. Therefore, higher dimensionality requires a higher grid resolution to achieve the same cost estimation accuracy within the limit of the generation time.

Based on these considerations, the following three grid plans were designed. The number of intervals (i.e., ten, five) in the grid plans is shown to be adequate.

- Grid plan 1 partitions the range of each model variable into *ten* equal intervals, and is designed for the range query and the kNN query, each of which has three model variables chosen. The resulting grid has the highest resolution of the three grid plans, and generates 1331 ($11 \times 11 \times 11$) training data points. This grid plan is not suitable for the window query, which has four model variables, because the training data set would have 14,641 ($11 \times 11 \times 11 \times 11$) data points at this grid resolution, which takes too long (>1 h) to generate (using the direct IO set-up in our experiments; see Section 6.3.4).

Fig. 10. Urban area map of Adams County in Pennsylvania State.

- Grid plan 2 is designed for the window query, which has four model variables, x_l, y_l, x_r, and y_r. It partitions the ranges of x_l and y_l into *ten* equal intervals, and the ranges of x_r and y_r into *five* equal intervals. The reason we can set the resolution of x_r and y_r to half that of x_l and y_r is that the ranges of x_r and y_r are smaller than the ranges of x_l and y_l, respectively, because $x_r > x_l$ and $y_r > y_l$. The resulting grid has intermediate resolution among the three, and generates 4356 ($11 \times 11 \times 6 \times 6$) training data points. This is a compromise between the need for more training data points owing to higher dimensionality and the need for fewer training data points owing to the limit on the generation time for the training data set.
- Grid plan 3 partitions the range of each model variable into *five* equal intervals, and can be used for any of the three spatial operators. The resulting grid has the lowest resolution of the three grid plans, and generates 216 ($6 \times 6 \times 6$) training data points for the range and kNN queries, and 1296 ($6 \times 6 \times 6 \times 6$) training data points for the window query.

Fig. 11(a) shows a training data set generated using grid plan 1. The query is a range query and the spatial data set is DS_4. Each arrow in the figure shows the disk IO cost incurred during query execution with the input argument *reference point* at the corresponding $\langle x, y \rangle$ and the input argument *distance* (d) set to 1000 m. The mesh in Fig. 11(b) shows a regression surface fitting the training data set.

6.3.3. Testing data sets

To evaluate a cost model, a testing data set is generated by taking random sample points in the model space. Then the spatial operator is executed at these sample points and the observed (i.e., actual) CPU and disk IO costs are collected. The estimated costs are subsequently computed using the model built from the training data set, and the relative errors are calculated from the actual and estimated costs.

6.3.4. Cost metrics and computing platform

The CPU cost is measured as the CPU time, and the disk IO cost is measured as the number of data pages fetched from disk to the main memory. All the experiments are performed using Oracle 9i Server running on SunOS 5.8 installed in Sun Ultra Enterprise 450 with four 300-MHz CPUs, 16-KB level 1 I-cache, 16-KB level 1 D-cache and 2 MB of level 2 cache per processor, 1024 MB of RAM, and 55 GB of hard disk. Oracle 9i was set up to use a 52-MB data cache and a 78-MB shared pool. The disk IO was set up to use *direct IO* for bypassing the OS buffer; this eliminates the unpredictable OS caching effects. Direct IO is commonly used in a DBMS to ensure recoverability by forcing direct page-writing to disk while bypassing the system buffer. This consumes significant time for disk IO-intensive operations, such as those performed in the experiments.

6.4. Experimental results

As mentioned in Section 6.2, the cost estimation accuracy is the primary performance factor in our experiments. Accuracy is reported in the form of an *error distribution*, i.e., the percentage of testing data points in

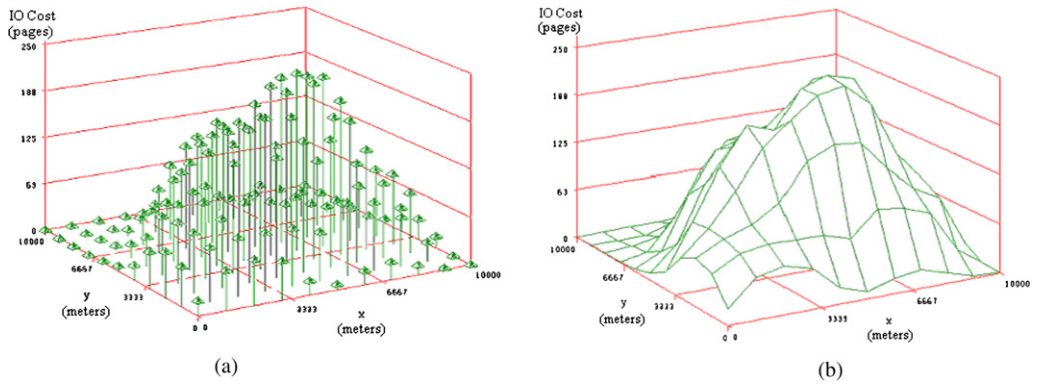

Fig. 11. Training data set and fitted surface for the disk IO cost for the range query (distance = 1000 m) using the synthetic data set DS_4. (a) Training data set (grid plan 1) and (b) fitted surface.

each range of the relative error, instead of the mean error. The reason for this is that the mean is biased because of some outliers with very high values. Most of these outliers stem from data points with very small (near zero) actual costs. A very small actual cost magnifies the relative error value significantly, even if the residual error (i.e., $|actual\ cost - estimated\ cost|$) is small. This happens frequently in the case of a spatial operator, especially if the spatial data set is sparse.

In this subsection, the error distributions for three experiments are presented: (1) for the three spatial operators; (2) for spatial data sets of different densities; and (3) for model training data sets of different sizes. In addition, the performance results for the secondary performance factors listed in Section 6.2 are reported.

6.4.1. Experiment 1: different spatial operators

Fig. 12 shows the results for the three spatial operators, obtained using two synthetic data sets (DS_3, DS_4) and the real data set (DS_5). The training data sets are of grid plan 1 for the range and kNN queries, and of grid plan 2 for the window query.

The figures show that, considering the CPU and disk IO costs together, the relative errors are less than 10% for at least ~70% of the testing points for the range and kNN queries, and at least ~45% of the testing points for the window query. Moreover, the relative errors are less than 20% for at least ~90% of the testing points for the range and kNN queries, and at least ~60% of the testing points for the window query. As reported by Gardarin et al. [30], 20% is a reasonable cost estimation error for query optimization. Moreover, if this result is compared with the errors (i.e., 29–222%) in an analytical cost model (of a kNN query with $k = 1$) shown in Section 3.1, it is apparent that the proposed method performs better, as it achieves less than 20% error for more than 90% of the testing points.

Fig. 12 shows that the window query incurs higher errors than the range query and the kNN query. This is largely because of the compromise made in designing grid plan 2. From the viewpoint of model precision, it is the density, not the number of training sample points—in other words, the grid resolution—that determines the precision of the model built using these training sample points. The training data set of the window query would have the same density as that of the range or kNN query if grid plan 1 had been used, and thus would have resulted in similar model precision. However, since the density of grid plan 2 is less than one-third the density of grid plan 1, naturally the error is higher.

6.4.2. Experiment 2: different spatial data-set densities

Only the synthetic data sets can be used in this experiment. Fig. 13 shows the results using two synthetic spatial data sets—the lower-density DS_2 and the higher-density DS_4—for each of the three spatial operators. The training data sets are of the same grid plans as those in Experiment 1.

The figures confirm that the relative error is higher for a lower-density spatial data set (see Cost estimation accuracy in Section 6.2). That is, in a sparse case, the actual cost is almost zero for many queries, and this causes the relative error to be very large, even if the residual error is very small. For example, if the estimated cost is 0.1 and the actual cost is 0.00001, then the relative error is 1,000,000% $(((0.1 - 0.00001)/0.00001) \times 100)$. The figures also show that the error gap between the two different densities is higher for the CPU cost than for the disk IO cost. This indicates that a significant proportion of the costs is incurred in the refinement step, which performs CPU-intensive geometric comparisons but little disk IO.

6.4.3. Experiment 3: model training data sets of different sizes

The results were compared using different grid plans, which lead to training data sets of different sizes. Fig. 14 shows the results for each of the three spatial operators. The larger training data set uses grid plan 1 for the range and kNN queries, and grid plan 2 for the window query; the smaller training data set uses grid plan 3 for all three spatial operators. The spatial data set used is the real data set (DS_5).

The figures show that, for the range and kNN queries, error gaps between the two grid plans are not as large as those for the window query. This indicates that the resolution of grid plan 3 (which has the lowest resolution) is already high enough to give accurate cost estimations for the queries with three model variables (i.e., range and kNN), but not for the query with four model variables (i.e., window).

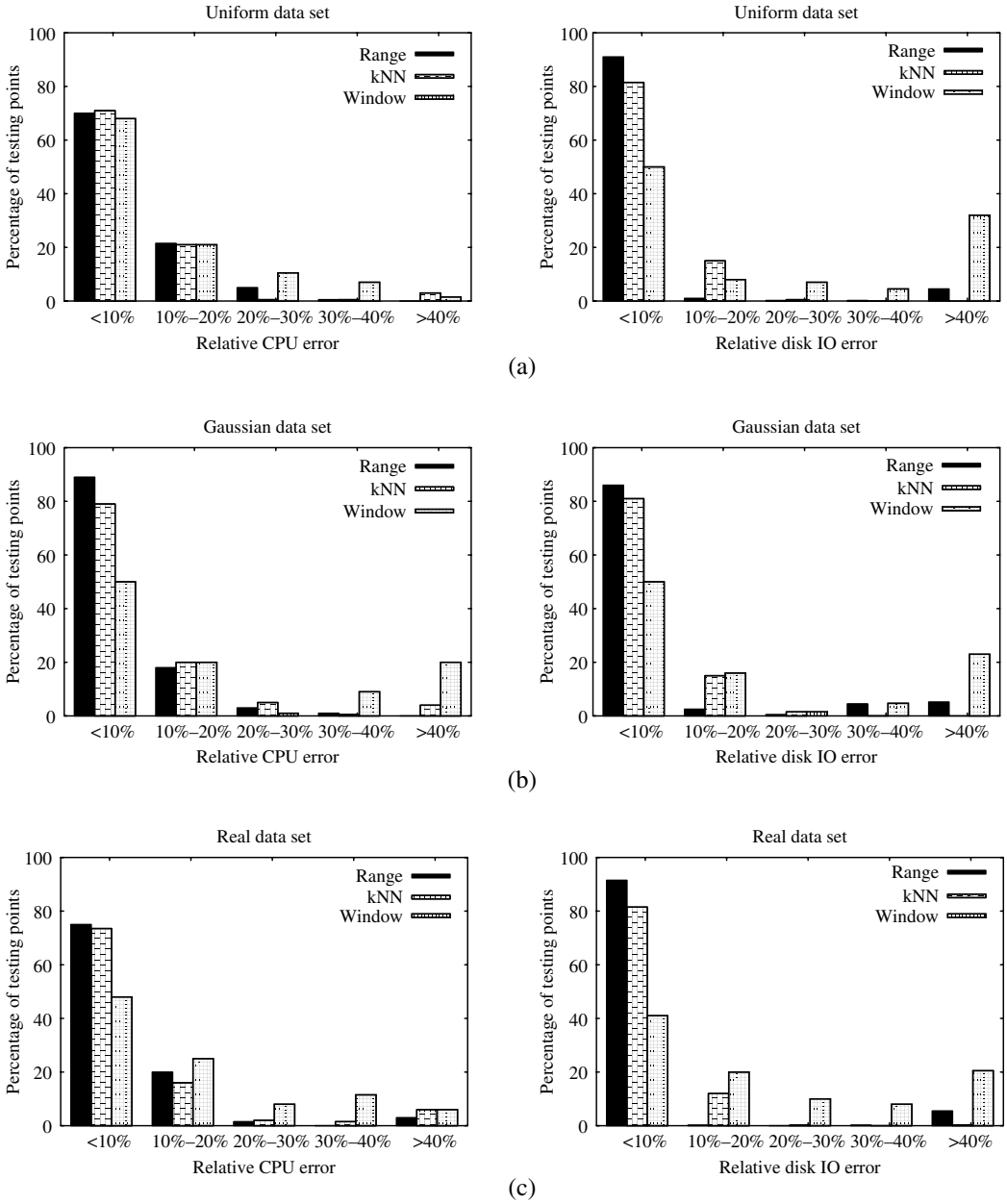

Fig. 12. Error distributions for different spatial operators. (a) Synthetic data set DS_3, (b) synthetic data set DS_4 and (c) real data set (DS_5). (Grid plan 1 for the range and kNN; plan 2 for window.)

6.4.4. Results for the secondary performance factors

The results for the other three performance factors are presented briefly here. They all indicate the feasibility of the proposed approach, especially considering the low computing power of the computing platform (Section 6.3.4) used in the experiments.

6.4.4.1. Cost estimation speed. Estimating both the CPU cost and the disk IO cost takes less than 50 ms on SAS for each spatial operator. (For each spatial operator, the costs were estimated 100 times using one testing data point each, and the average estimation time was calculated.) Considering that SAS programs are *interpreted* at

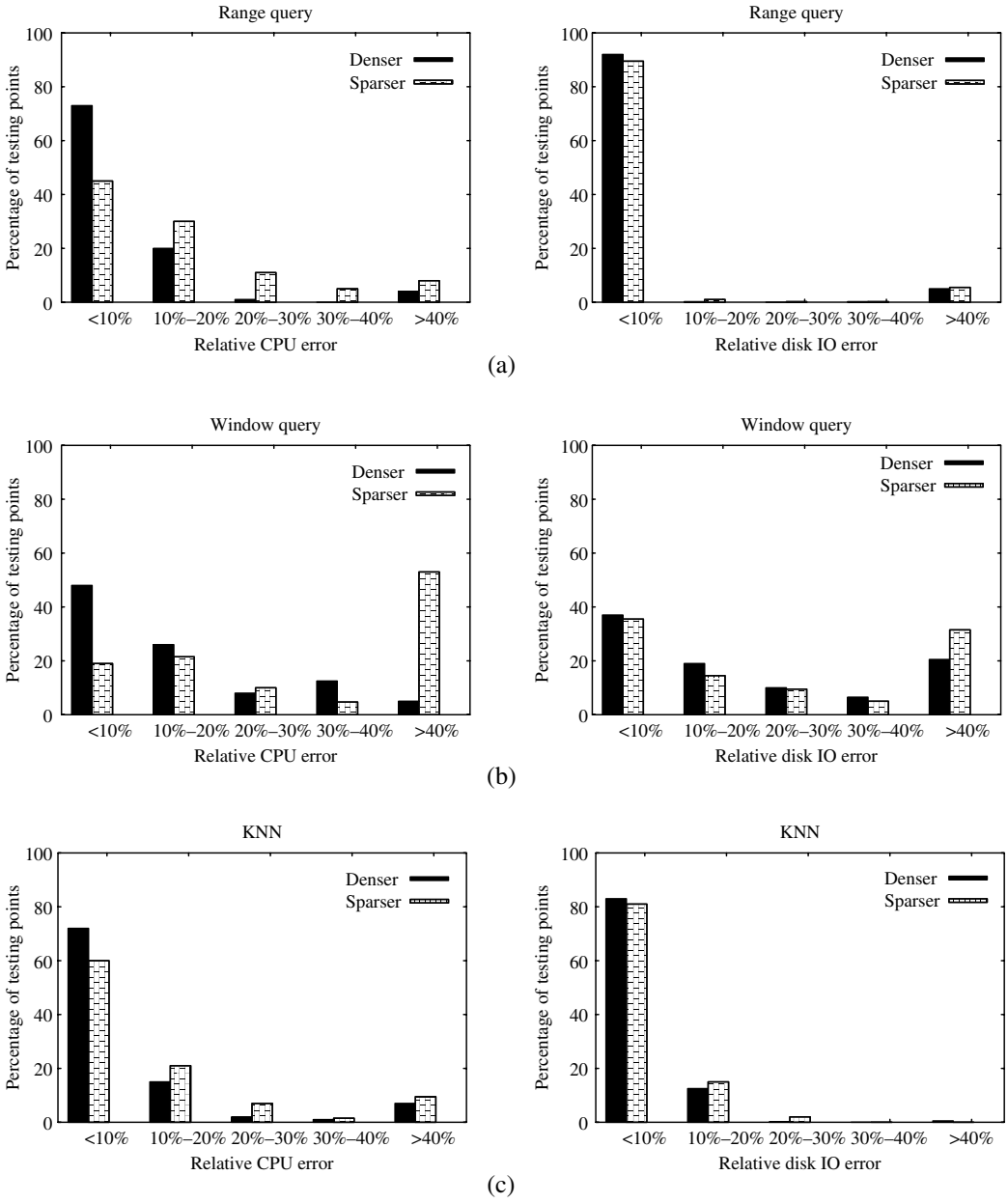

Fig. 13. Error distributions for different spatial data-set densities (denser, DS_4; sparser, DS_2). (a) Range query, (b) window query and (c) k-nearest neighbor query. (Grid plans: plan 1 for the range and kNN; plan 2 for the window.)

run-time, and are thus slower than running compiled programs, and that a low-end computing platform has been used, the actual cost estimation speed after incorporating this approach into a DBMS query optimizer should be fast enough.

6.4.4.2. Model building speed. The model building took 0.5–1 min for 1331 training points (i.e., grid plan 1) for the range and the kNN queries, and 2–4 min for 4356 training points (i.e., grid plan 2) for the window query. These overheads are trivial, since the model building is carried out off-line.

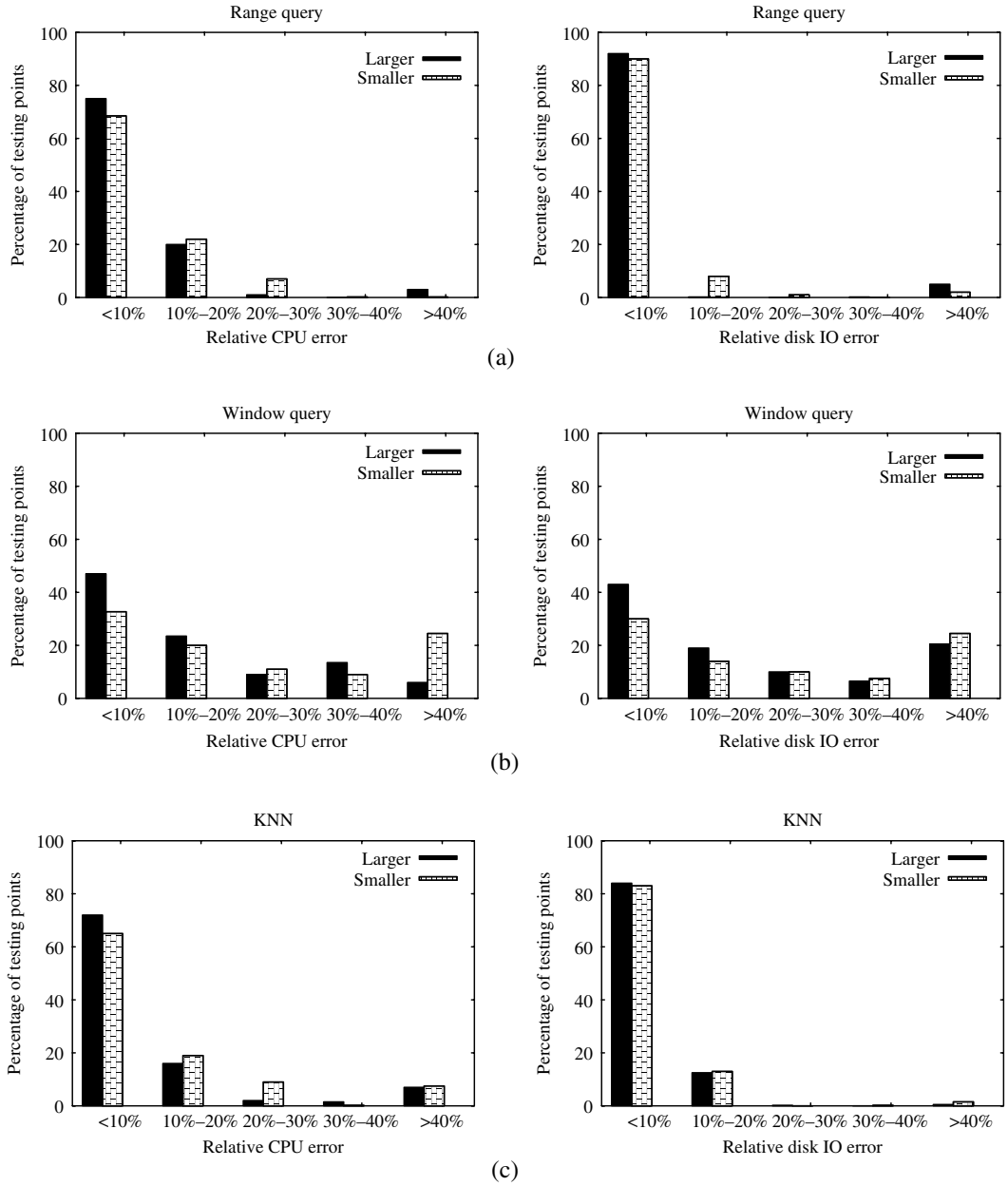

Fig. 14. Error distributions for model training data sets of different sizes (using DS_5). (a) Ranger query (larger, grid plan 1; smaller, grid paln 3), (b) window query (larger, grid plan 2; smaller, grid plan 3) and (c) *k*-nearest neighbor query (larger, grid plan 1; smaller, grid plan 3).

6.4.4.3. Generation speed for training data set. For the range and the kNN queries when using grid plan 1, training data set generation took 1–2 min on the sparser synthetic data sets (i.e., DS_1, DS_2), 10–15 min on the denser synthetic data sets (i.e., DS_3, DS_4), and 35–50 min on the real data set. For the window query, it took 20–32 min for grid plan 2 on the denser synthetic data sets. These are all shorter than 1 h. Note that it took significantly longer for the real data set than for the synthetic data sets. This is because of the higher

disk IO cost resulting from two factors: the real data set has more objects—15,000 compared with 1000 (in DS_1 and DS_2) and 10,000 (in DS_3 and DS_4)—and requires more disk pages to store them.

7. Conclusion

In this paper, a statistical approach for building a cost model for a spatial operator was presented. Given model variables of a spatial operator and a grid plan, the system generates a training data set. This is accomplished by repeatedly executing the spatial operator using input argument values corresponding to the sampled values of model variables. Then cost models of the CPU and disk IO costs are built separately by fitting the training data using non-parametric regression.

The model building procedure used in this approach was described and then applied to the range query, window query, and k-nearest neighbor query. Based on the information that each retrieves, the model variables were determined for the spatial operators. A cost model was then built for each spatial operator and the model performance was evaluated in terms of the relative errors in cost estimation.

In experiments, three model variables were used for the range and k-nearest neighbor queries, and four for the window query. The results show that more than 90% of the testing points have relative errors of less than 20% for the range and k-nearest neighbor queries, and more than 60% of the testing points for the window query (for which the training data sets have limited sizes to limit their generation times). In addition, the results show lower relative errors for denser spatial data sets and/or larger training data sets. The speed of on-line cost estimation, off-line model building, and off-line generation of the training data-set indicates that this approach is suitable for incorporation into a real DBMS.

The main advantages of this approach are (1) the ease of use for ordinary database users and (2) the high precision of the cost model built, attributed to the flexible modeling power of non-parametric regression. The disadvantage is the overhead of re-training the model if frequent updates occur in the spatial database. We believe that this method can replace the complex analytical approach in many situations for which the re-training overhead is not prohibitive.

The spatial operators used in experiments are types of spatial selection and aggregate operators. Future work will include the cost modeling of other types of spatial operators, such as spatial *joins* [62] and the k closest pairs [18]. It is anticipated that this will be a straightforward extension of the current work. For example, a spatial join overlap (S_1, S_2), where S_1 and S_2 are spatial data sets, finds all objects overlapping between S_1 and S_2. The model variables should include the cardinality of S_1 and the cardinality of S_2, because the costs should increase with cardinality. The k closest pairs, $\mathrm{kCP}(S_1, S_2)$, finds the k closest pairs of objects from the two spatial data sets. It is expected that such a cost model is an extension of that for kNN queries.

Other future work will involve combining the analytical and statistical approaches. This will involve dividing query processing into smaller steps and building the analytical cost model of each step. The resulting cost models for all steps will then be integrated into one statistical model.

Acknowledgements

We thank the Computing Staff at the College of Engineering and Mathematics of the University of Vermont for their system support, including setting up and maintaining the Oracle 9i Enterprise Server™ and Oracle Spatial™ . We also thank Li Chen for acquiring and setting up the real data sets used in the experiments. Our special thanks go to the anonymous reviewers for their considerate and elaborate comments, which were invaluable to improve the original manuscript. We also thank Professor Pedrycz, the Editor-in-Chief, for his help to improve the linguistic quality of the manuscript. This paper is based upon work supported by the National Science Foundation under Grant No. IIS-0415023 and by the Department of Energy under Grant No. DE-FG02-ER45962.

References

[1] A. Aboulnaga, J. Naughton, Accurate estimation of the cost of spatial selections, in: Proceedings of the 16th IEEE International Conference on Data Engineering, San Diego, CA, 2000, pp. 123–134.

[2] S. Acharya, V. Poosala, S. Ramaswamy, Selectivity estimation in spatial database, in: Proceedings of the ACM SIGMOD International Conference on Management of Data, Philadelphia, PA, 1999, pp. 13–24.
[3] W. Aref, H. Samet, Optimization strategies for spatial query processing, in: Proceedings of the 17th International Conference on Very Large Data Bases, Barcelona, Spain, 1991, pp. 81–90.
[4] S. Arya, D.M. Mount, N.S. Netanyahu, R. Silverman, A. Wu, An optimal algorithm for approximate nearest neighbor searching fixed dimensions, Journal of ACM 45 (6) (1998) 891–923.
[5] C. Baptista, A geolibrary for multimedia data sets: design and implementation issues, in: Proceedings of the ACM symposium on Applied computing, Madrid, Spain, 2002, pp. 488–492.
[6] N. Beckmann, H. Kriegel, R. Schneider, B. Seeger, The R*-tree: an efficient and robust access method for points and rectangles, in: Proceedings of the ACM SIGMOD International Conference on Management of Data, Atlantic City, NJ, 1990, pp. 322–331.
[7] A. Belussi, C. Faloutsos, Estimating the selectivity of spatial queries using the 'correlation' fractal dimension, in: Proceedings of the 21st International Conference on Very Large Data Bases, Zurich, Switzerland, 1995, pp. 299–310.
[8] S. Berchtold, C. Böhm, D. Keim, H. Kriegel, A cost model for nearest neighbor search in high-dimensional data spaces, in: Proceedings of the ACM SIGACT–SIGMOD–SIGART Symposium on Principles of database systems, New York, NY, 1997, pp. 78–86.
[9] S. Berchtold, D. Keim, H. Kriegel, The X-tree: an index structure for high-dimensional data, in: Proceedings of the International Conference on Very Large Data Bases, Bombay, India, 1996, pp. 28–39.
[10] J. Bishop, Urban Areas of Counties in the Pennsylvania State, Pennsylvania State University, University Park, PA, 1998. Available from: <www.pasda.psu.edu/access/urban.shtm>.
[11] J. Boulos, K. Ono, Cost estimation of user-defined methods in object-relational database systems, SIGMOD Record 28 (3) (1999) 22–28.
[12] C. Böhm, A cost model for query processing in high dimensional data spaces, ACM Transactions on Database Systems 25 (2) (2000) 129–178.
[13] C. Böhm, S. Berchtold, D. Keim, Searching in high-dimensional spaces – index structures for improving the performance of multimedia databases, ACM Computing Surveys 33 (3) (2001) 322–373.
[14] F. Bookstein, Principal warps: thin plate splines and the decomposition of deformations, IEEE Transactions on Pattern Analysis and Machine Intelligence 11 (1989) 567–585.
[15] T. Brinkhoff, H. Kriegel, R. Schneider, B. Seeger, Multi-step processing of spatial joins, in: Proceedings of the ACM SIGMOD Inernational Conference on Management of Data, Minneapolis, MN, 1994, pp. 237–246.
[16] N. Cenzov, Evaluation of an unknown distribution density from observations, Soviet Mathematical Doklady 3 (1962) 1559–1562.
[17] Y. Chen, Y. Hung, C. Fuh, Fast algorithm for nearest neighbor search based on a lower bound tree, in: Proceedings of the IEEE International Conference on Computer Vision, Vancouver, Canada, 2001, pp. 446–453.
[18] A. Corral, Y. Manolopoulos, Y. Theodoridis, M. Vassilakopoulos, Closest pair queries in spatial databases, in: Proceedings of the ACM SIGMOD International Conference on Management of Data, Dallas, TX, 2000, pp. 189–200.
[19] R. deSilva, K. Wood, J. Beaman, An algebraic approach to geometric query processing in CAD/CAM applications, in: Proceedings of the ACM Symposium on Solid Modeling Foundations and CAD/CAM Applications, Austin, Texas, 1991, pp. 73–86.
[20] W.S. Cleveland, Robust locally-weighted regression and smoothing scatterplots, Journal of American Statistical Association 74 (1979) 829–836.
[21] W. Cleveland, E. Grosse, Computational methods for local regression, Statistics and Computing 1 (1991) 47–62.
[22] M. Egenhofer, Spatial SQL: a query and presentation language, IEEE Transactions on Knowledge and Data Engineering 6 (1) (1994) 86–95.
[23] C. Faloutsos, I. Kamel, Beyond uniformity and independence: analysis of R-trees using the concept of fractal dimension, in: Proceedings of the ACM SIGACT–SIGMOD–SIGART Symposium on Principles of Database Systems, Minneapolis, MN, 1994, pp. 4–13.
[24] C. Faloutsos, T. Sellis, N. Roussopoulos, Analysis of object oriented spatial access methods, in: Proceedings of the ACM SIGMOD International Conference on Management of Data, San Francisco, CA, 1987, pp. 426–439.
[25] J. Fox, Nonparametric Simple Regression: Smoothing Scatterplots, Sage Publications, Thousand Oaks, California, 2002.
[26] J. Friedman, F. Baskett, L. Shustek, An algorithm for finding nearest neighbors, IEEE Transactions on Computers 24 (10) (1975) 1000–1006.
[27] J. Friedman, J. Bentley, R. Finkel, An algorithm for finding best matches in logarithmic expected time, ACM Transactions on Mathematical Software 3 (3) (1977) 209–226.
[28] K. Fukunaga, P. Narendra, A branch and bound algorithm for computing *k*-nearest neighbors, IEEE Transactions on Computers 24 (1975) 750–753.
[29] V. Gaede, O. Günther, Multidimensional access methods, Computing Surveys 30 (2) (1998) 170–231.
[30] G. Gardarin, F. Sha, Z.-H. Tang, Calibrating the query optimizer cost model of IRO-DB: an object-oriented federated database system, in: Proceedings of the 22nd International Conference on Very Large Data Bases, Bombay, India, 1996, pp. 378–389.
[31] L. Gorden, R.A. Olshen, Consistent nonparametric regression from recursive partioning schemes, Journal of Multivariate Analysis 10 (1980) 611–627.
[32] R. Guting, An introduction to spatial database systems, VLDB Journal 3 (2004) 357–399.
[33] A. Guttman, R-trees: a dynamic index structure for spatial searching, in: Proceedings of the ACM SIGMOD International Conference on Management of Data, Boston, MA, 1984, pp. 47–57.

[34] R. Helm, K. Marriott, M. Odensky, Constraint-based query optimization for spatial databases, in: Proceedings of the ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, Denver, CO, 1991, pp. 181–191.
[35] P. Huber, Projection pursuit, Annals of Statistics 13 (1985) 435–475.
[36] N. Katayama, S. Satoh, The SR-tree: an index structure for high-dimensional nearest neighbor queries, in: Proceedings of the ACM SIGMOD International Conference on Management of Data, Tucson, AZ, 1997, pp. 368–380.
[37] B. Kim, S. Park, A fast k nearest neighbor finding algorithm based on the ordered partition, IEEE Transactions on Patten Analysis and Machine Intelligence 8 (1986) 761–766.
[38] B. Kobler, J. Berbert, NASA earth observing system data information system (EOSDIS), in: Digest of Papers: 11th IEEE Symposium on Mass Storage Systems, Monterey, California, 1991, pp. 18–19.
[39] F. Korn, B. Pagel, C. Faloutsos, On the "dimensionality curse" and the "self-similarity blessing", IEEE Transactions on Knowledge and Data Engineering 13 (1) (2001) 96–111.
[40] H. Kriegel, P. Kunath, M. Pfeifle, M. Renz, Statistic driven acceleration of object-relational spatial index structures, in: Proceedings of the International Conference on Database Systems for Advanced Applications, Jeju Island, Korea, 2004, pp. 169–183.
[41] H. Kriegel, P. Knuath, M. Pfeifle, M. Pötke, M. Renz, P.-M. Straus, Stochastic driven relational R-tree, in: Brazilian Symposium on Geoinformatics, São Paulo, Brazil, 2003. Avaolabel from: <www.geoinfo.info/geoinfo2003/papers/geoinfo2003-19.pdf>.
[42] H. Kriegel, A. Muller, M. Potke, T. Seidl, Spatial data management for computer-aided design, in: Proceedings of the ACM SIGMOD International Conference on Management of Data, Santa Barbara, CA, 2001, p. 614.
[43] R. Lanzelotte, P. Valduriez, M. Zait, On the effectiveness optimization search strategies for parallel execution spaces, in: Proceedings of the 19th International Conference on Very Large Data Bases, Dublin, Ireland, 1993, pp. 493–504.
[44] B. Lee, V. Kannoth, J. Buzas, Statistical cost-modeling of financial time series functions, Journal of Computers and Applications, in press.
[45] K. Lin, H. Jagadish, C. Faloutsos, The TV-tree: an index structure for high-dimensional data, VLDB Journal 3 (1994) 517–542.
[46] D. Loftsgaarden, G. Queensberry, A nonparametric estimate of a multivariate density function, Annals of Mathematics and Statistics 36 (1965) 1049–1051.
[47] J. Meinguet, Multivariate interpolation at arbitrary points made simple, Journal of Applied Mathematical Physics 30 (1979) 292–304.
[48] A. Nanopoulos, Y. Theodoridis, Y. Manolopoulos, An efficient and effective algorithm for density biased sampling, in: Proceedings of the International Conference on Information and Knowledge Management, McLean, VA, 2002, pp. 63–68.
[49] J. Orenstein, F. Manola, Probe spatial data modeling and query processing in an image database application, IEEE Transactions on Software Engineering 14 (5) (1998) 611–629.
[50] B. Pagel, H. Six, H. Toben, P. Widmayer, Towards an analysis of range query performance in spatial data structures, in: Proceedings of the ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, Washington, DC, 1993, pp. 214–221.
[51] A. Papadopoulos, Y. Manolopoulos, Performance of nearest neighbor queries in R-trees, in: Proceedings of the International Conference on Database Theory, Delphi, Greece, 1997, pp. 394–408.
[52] F. Rao, L. Zhang, X. Yu, Y. Li, Y. Chen, Spatial hierarchy and OLAP-favored search in spatial data warehouse, in: Proceedings of the International Workshop on Data Warehousing and OLAP, New Orleans, LA, 2003, pp. 48–55.
[53] H. Reinsch, Smoothing by spline functions, Numerische Mathematik 10 (1967) 177–183.
[54] E. Riedel, G. Gibson, C. Faloutsos, Active storage for large-scale data mining and multimedia, in: Proceedings of the 24th International Conference on Very Large Data Bases, New York, NY, 1998, pp. 62–73.
[55] J.T. Robinson, The K-D-B-tree: a search structure for large multidimensional dynamic indexes, in: Proceedings of the ACM SIGMOD International Conference on Management of Data, Ann Arbor, MI, 1981, pp. 10–18.
[56] M. Rosenblatt, Remarks on some nonparametric estimates of a density function, Annals of Mathematics and Statistics 27 (1956) 642–669.
[57] N. Roussopoulos, S. Kelley, F. Vincent, Nearest neighbor queries, in: Proceedings of the ACM SIGMOD International Conference on Management of Data, San Jose, CA, 1995, pp. 71–79.
[58] D. Rupert, M. Wand, R. Carroll, Semiparametric Regression, Cambridge University Press, Cambridge, United Kingdom, 2003.
[59] T. Sellis, N. Roussopoulos, C. Faloutsos, The R+-tree: a dynamic index for multi-dimensional objects, in: Proceedings of the 13th International Conference on Very Large Data Bases, Brighton, England, 1987, pp. 507–518.
[60] S. Shekhar, S. Chawla, Spatial Databases: a Tour, Prentice-Hall, Upper Saddle River, NJ, 2003.
[61] C. Stone, Consistent nonparametric regression (with discussion), Annals of Statistics 5 (1977) 595–645.
[62] Y. Theodoridis, E. Stefanakis, T. Sellis, Efficient cost models for spatial queries using R-trees, IEEE Transactions on Knowledge and Data Engineering 12 (1) (2000) 19–32.
[63] W. Wang, J. Yang, R. Muntz, PK-tree: a spatial index structure for high dimensional point data, in: Information Organization and Databases: Foundations of Data Organization, Kluwer Academic Publishers, Norwell, Massachusetts, 1998, pp. 281–293.
[64] G. Wahba, Spline models for observational data, CBMS-NSF Regional Conference Series in Applied Mathematics, number 59, Society for Industrial & Applied Mathematics (SIAM), Philadelphia, Pennsylvania, 1998.
[65] G. Wahba, J. Wendelberger, Some new mathematical methods for variational objective analysis using splines and cross-validation, Monthly Weather Review 108 (1980) 1122–1145.
[66] G. Watson, Smooth regression analysis, Sankhya – The Indian Journal of Statistics 26 (1964) 359–372.

[67] G. Whaba, Optimal convergence properties of variable knot, kernel, and orthogonal series methods for density estimation, Annals of Statistics 3 (1975) 15–29.
[68] C. Yu, B. Ooi, K. Tan, H. Jagadish, Indexing the distance: an efficient method to KNN processing, in: Proceedings of the International Conference on Very Large Data Bases, Roma, Italy, 2001, pp. 421–430.
[69] Open GIS consortium, Open GIS simple feature specification for SQL (revision 1.1), OpenGIS Project Document 99-049, May 5, 1999. Available from: <www.opengeospatial.org/docs/99-049.pdf>.
[70] SAS OnlineDoc: SAS/STAT, SAS Institute Inc., Cary, NC, 2004. Available from: <v8doc.sas.com/sashtml/>.