APLA: Indexing Arbitrary Probability Distributions

Vebjorn Ljosa and Ambuj K. Singh
University of California, Santa Barbara
{ljosa,ambuj}@cs.ucsb.edu

Abstract

The ability to store and query uncertain information is of great benefit to databases that infer values from a set of observations, including databases of moving objects, sensor readings, historical business transactions, and biomedical images. These observations are often inexact to begin with, and even if they are exact, a set of observations of an attribute of an object is better represented by a probability distribution than by a single number, such as a mean.

In this paper, we present adaptive, piecewise-linear approximations (APLAs), which represent arbitrary probability distributions compactly with guaranteed quality. We also present the APLA-tree, an index structure for APLAs. Because APLA is more precise than existing approximation techniques, the APLA-tree can answer probabilistic range queries twice as fast. APLA generalizes to multiple dimensions, and the APLA-tree can index multivariate distributions using either one-dimensional or multi-dimensional APLAs.

Finally, we propose a new definition of k-NN queries on uncertain data. The new definition allows APLA and the APLA-tree to answer k-NN queries quickly, even on arbitrary probability distributions. No efficient k-NN search was previously possible on such distributions.

1 Introduction

Databases of uncertain values have received much recent attention because querying uncertain values is technically challenging and benefits a variety of applications, ranging from moving object databases and sensor networks to biomedical and other scientific databases.

A database of moving objects, such as taxicabs, police cars, delivery trucks, convicts, or ships, will contain uncertain locations not only because the position is inexactly measured in the first place, but also because radio range, energy consumption, and cost make it infeasible to continuously update the database with the object's current location. We must therefore be able to answer queries based on a probability density function (pdf) of the object's predicted position. The pdf can be as simple as a uniform density within a range determined from the object's max. speed, or it can be an arbitrarily complex function derived from models of the road network, traffic conditions, and the object's past behavior.

Networks of power-constrained sensors that measure temperature, atmospheric pressure, chemical concentrations or other quantities generate uncertain values for the same reasons as moving object databases. They also introduce another level of uncertainty because measurements are aggregated in space and time: If a region contains multiple sensors, then the aggregate temperature of that region is a pdf—even if each measurement is exact.

The combination of high-throughput data acquisition techniques and new search technology has transformed genetics and proteomics into data-driven fields: Querying the wealth of existing data has become both a source of new hypotheses and a means of testing them. Several large projects are underway to make data-driven research a reality in other biomedical fields [12, 17], and this pursuit has been a rich source of new challenges for the database community. Microscopy is a cornerstone of biomedical research, and microscopy images and the observations or measurements from the images are at the very core of biomedical databases. Examples include the density of cells in a certain tissue or the thickness of a certain layer of neurons in the retina. The measurements are numerous: the thickness of a layer of neurons in the retina can, for instance, be measured in hundreds of places in each image, yielding different numbers. The data are a mix of the results of image analysis programs and manual observations by scientists. There is an element of uncertainty associated with every value, not only because both analysis programs and humans make mistakes, but also because image resolution, focus, and noise limit the accuracy of measurements.

Queries ask for summaries of, or conclusions drawn from, the observations or measurements. A query may ask, for instance, for all tissue samples that have cell densities in a certain range. Note that there may be hundreds of measurements of cell density in different regions of each sample, but the query asks for the samples, expecting the system to abstract away from the individual measurements. It is obviously not a satisfactory solution just to find all samples that have at least one measurement in the query region, for this will return many samples that are irrelevant because only a few of their measurements were in the range. An attractive solution is to store the summary as a probabilistic value, i.e., a probability density function (pdf) over measured quantities. For the cell density example, this representation could capture that the density is $\sim$32,000 cells/mm^2 most places but $\sim$28,000 cells/mm^2 in about 20 % of the sample. In simple cases, the summary can be obtained by computing a histogram of the measurements, but it can also be the result of complex analyses involving measurements of different quantities.

Uncertain values result not only from summarizing imprecise scientific measurements, but wherever a large number of records are grouped and treated as one object for the purpose of search—even in otherwise mundane business applications. As an example, imagine the database behind a car auction website. All previous sales are stored in the database, and serve as a basis for users to learn what models they can expect to get for what price. A user

that asks "How much do 1992 Honda Civics sell for?" expects not a single number (e.g., a mean), but information such as "usually $3000–4000, but sometimes as low as $2000." The system can give such an answer by returning the pdf of the attribute *price* of the object *1992 Honda Civic*. Other examples of probabilistic queries include "Find all models that usually sell for between $20,000 and $30,000" (a range query) and "Find 10 models that sell for around $25,000" (a k-NN query).

The database challenge, then, is to answer queries on pdfs. This has been studies by several authors [2, 3, 4, 6], but under the assumption that the pdfs are uniform or Gaussian. These assumptions allow for interesting and efficient index structures, and can be appropriate for the uncertainty of many of the individual measurements. They are too restrictive for pdfs that occur as a result of summarization, however, for the observations being summarized may be generated by different mechanisms. For instance, a summary of the density of bipolar cells in a detached (injured) retina will have two peaks, corresponding to parts of the retina that are injured and healthy, respectively. Fitting a model distribution, such as a Gaussian, to the data is only appropriate when the data are well understood, and in a scientific database, the most interesting data to query are precisely the ones that are *not* well understood.

We are therefore concerned with probabilistic queries on *arbitrary* pdfs in this paper. The major contributions are as follows:

- *Adaptive, piecewise-linear approximations (APLAs), compact representations of arbitrary pdfs.* Because APLAs have guaranteed error bounds, they can be used to prune objects during search, and because they are better approximations that existing techniques, they can prune more objects and thereby answer queries faster.

- *APLA-tree, an index structure for APLAs.* Because APLAs can approximate a set of other APLAs, we can build a tree structure, prune entire subtrees during search, and thereby scale to large datasets.

- *A novel definition of k-NN queries on uncertain data.* No efficient solution existed for k-NN search on arbitrary distributions, but the new definition allows such distributions to be indexed with APLA and the APLA-tree; consequently, k-NN queries can be answered quickly.

The remainder of the paper is organized as follows. In Section 2, we investigate probabilistic range queries and discuss desiderata for indexing schemes. In Section 3, we suggest a novel definition for k-NN queries on pdfs. Section 4 proposes APLA and the APLA-tree index structure. Section 5 evaluates the techniques experimentally before Section 6 concludes the paper.

2 Range queries on uncertain data

Let $f_i : \mathbb{R}^d \mapsto [0,1]$ be the joint probability density function of d real-valued attributes of an object o_i in the database. For any point $\vec{x} = [x_1, \ldots, x_d]$, $f_i(\vec{x})$ is the probability that a random observation of the object will have the value x_1 for o_i's first attribute, the value x_2 for x_i's second attribute, and so on.

A probabilistic range query [4, 19] consists of a query range $R \subseteq \mathbb{R}^d$ and a probability threshold τ. It returns all objects for which a random observation has a probability of at least τ of being in the query range, i.e.,

$$Q(R,\tau) = \{o_i : \int_R f_i(\vec{x})\,\mathrm{d}\vec{x} \geq \tau\}.$$

The probability that a random observation appears in the query range is called the object's *appearance probability*. We limit our investigation to rectilinear query ranges. (A query with arbitrary-shaped range can be answered by querying with the MBR of the range, then refining the results.)

If the pdf is represented by a histogram, an upper bound of the appearance probability can be computed by adding up the depths of all bins that intersect the query range, and the upper bound can be turned into a lower bound by subtracting the bins that intersect the boundary of the query range. This is inefficient, however: assuming that the query range intersects the same number of bins, m, in each dimension, the sum has m^d terms.

Computing the sum can be avoided by turning the pdf $f_i(\vec{x})$ into a cumulative distribution function (cdf). The cdf $F_i(\vec{x})$ is the probability that a random observation will have a value $\vec{x'}$ that is element-wise smaller than $\vec{x}$. By applying the inclusion–exclusion theorem, the appearance probability can now be computed as a sum of 2^d terms. As an example in 2-space, if R is the rectangle defined by the points (x_0, y_0) and (x_1, y_1), the appearance probability of an object o_i is $F_i(x_1,y_1) - F_i(x_0,y_1) - F_i(x_1,y_0) + F_i(x_0,y_0)$.

Using the cdf reduces the amount of computation, but the I/O cost is still unacceptable because detailed multidimensional histograms are bulky and because we have to inspect one such histogram for every object. A different representation of the cdf is therefore needed. A good representation should have the following three properties:

1. It should be compact in size,

2. It should give tight lower and upper bounds for the cdf at every point, and

3. It should be able to summarize the representations of a set of objects, so that the bounds computed from the summary are valid bounds for every object in the set.

The third property is what allows a representation to be used in search trees: During search, entire subtrees can be pruned because it can be determined from the compact summary stored in an internal node that none of the objects in the subtree satisfies the query.

2.1 Previous solutions

If the bulk of the histograms is the source of the problem, an obvious remedy is to use histograms with fewer bins. How to find a compact histogram that is a good match to the original pdf is less obvious, however. Jagadish et al. [8] give a dynamic programming algorithm that partitions a one-dimensional histogram into a given number of intervals so as to minimize the sum of squared difference between the mean over an interval and any value within that interval. The algorithm can also be adapted to minimize the maximal absolute difference. Jagadish et al.'s algorithm elegantly reduces the size of one-dimensional histograms, but in two or more dimensions, the problem of whether a given partitioning is optimal is NP-complete, so heuristics must be used [15].

The fact that a histogram is a piecewise-*constant* approximation is cause for concern. A piecewise-constant approximation

may be as good as any when approximating arbitrary functions, but the cdf is a special case because it is monotonically increasing from 0 to 1. The approximation will therefore be good in the middle of each interval being approximated by a constant function but not so good closer to the ends of the interval. This raises the question of whether something else than a constant function could provide a better approximation.

If a piecewise-constant function is not such a good approximation for the cdf, why not approximate the pdf instead, and then compute the cumulative of the result? This approach, described by Jagadish et al. [8], has the opposite problem: not only the approximate function accumulates, but also its error, so the resulting approximate cdf will be a good fit close to the bin boundaries and bad fit in the middle of a bin. The question remains of whether a different approximation would be better.

Keogh et al. [9] use piecewise-constant approximations on time series (where the objections just discussed do not apply). Each time series is approximated by s constant pieces, then inserted into an R-tree as a $2s$-dimensional point. (Odd dimensions contain the constant value of each piece, even dimensions the boundaries between pieces.) The MINDIST distance function of the R-tree is then modified so that time series can be searched by similarity.

Korn et al.'s OptimalSplines [10] fit a series of B-splines by maximum likelihood. This works well for selectivity estimation, where the expected error over many queries is what matters: It is acceptable for small parts of the domain to be poorly approximated because it will only affect some queries. We are interested, however, in tight upper and lower bounds so we can prune objects (or entire subtrees). OptimalSplines are not appropriate here because the maximal error is what is important for the bounds' tightness: If the approximation is poor even in a small part of the domain, the bounds become loose and the pruning power is diminished for all queries. We experiment with OptimalSplines in Section 5.

Tao et al.'s conservative functional boxes (CFBs) [19] treat each dimension separately and envelope the cdf with four lines, found by linear programming. The first two lines bound the part of the cdf where $F_i(x) \leq 0.5$ from above and below, respectively. The other two bound the part of the cdf where $F_i(x) \geq 0.5$. Always using $F_i(x) = 0.5$ as a boundary is both a blessing and a curse: It results in a compact representation because the partitioning point does not have to be stored, but the bounds can be very loose because the partitioning point does not adapt to the data. In Section 5 we conduct experiments that compare our proposed techniques to CFBs and the U-tree index structure based on them.

3 k-NN queries on uncertain data

It is not obvious how to define k-NN queries on probability distributions. What exactly does it mean for one distribution to be "nearer" than another to a query point $\vec{q}$?

Cheng et al. [3] suggested a definition for 1-NN queries based on the probability p_i that a random observation of an object o_i is closer to $\vec{q}$ than random observations from all other objects in the database. The object with the highest p_i is considered $\vec{q}$'s nearest neighbor. The probability p_i is defined as

$$p_i = \int_0^\infty P(d(\vec{q},o_i) = r) \prod_{j \neq i} P(d(\vec{q},o_j) > r)\,\mathrm{d}r, \tag{1}$$

where d is any distance measure. Cheng et al. develop efficient solutions for the special cases where the pdf is a uniform distribution constrained to line segments and circles. However, for arbitrary distributions, computing p_i is inefficient and impractical because p_i depends not only on o_i's pdf, but also the pdfs of all other objects in the database. This is in stark contrast to k-NN queries on deterministic data, where data points are ranked by their distance to the query point—a quantity that is easily computed for each data point without reference to the rest of the database.

As an alternative, we propose to define probabilistic k-NN queries so they return the k objects that have the smallest *expected distance* (ED) from the query point $\vec{q}$, i.e.,

$$\mathrm{ED}(\vec{q},o_i) = \int_{\mathbb{R}^d} d(\vec{q},\vec{x}) f_i(\vec{x})\,\mathrm{d}\vec{x}. \tag{2}$$

$\mathrm{ED}(\vec{q},o_i)$ is also known as f_i's first moment about $\vec{q}$, and can be computed solely from the query point $\vec{q}$ and o_i's pdf, f_i.

This definition is not equivalent to Cheng et al.'s definition (consider a 1-NN query with query point 0, two objects with pdfs defined by the histograms $[0.30, 0.02, 0.68]$ and $[0.08, 0.41, 0.51]$, respectively, and an L_p-norm as distance measure), but we believe it is an equally natural interpretation of a query that asks for the k distributions that are "nearest" a particular query point. As an example, Figure 1 plots the pdf for a mixture of Gaussians and the portion of $\mathrm{ED}(q)$ that is inside the uncertainty region. (The *uncertainty region* of an object is a closed region such that object's pdf in nonzero only inside the region [3].)

When the distance function d is the L_1 ("Manhattan") distance, ED has an important property, namely that if $\vec{q}$ is outside the uncertainty region of o_i, then $\mathrm{ED}(\vec{q},o_i)$ is the distance from $\vec{q}$ to the mean of o_i, as shown by the following lemma.

Lemma 1. *If $\vec{q}$ is outside the uncertainty region of o_i, then*

$$\mathrm{ED}(\vec{q},o_i) = \sum_{k=1}^{d} w_k |q_k - \mu_{ik}| = \mathrm{L}_1(\vec{q},\mu_i),$$

where w_k is the weight of the j-th dimension (1 if unweighted L_1-distance is used) and μ_{ij} is the j-th dimension of the mean of f_i.

Proof. Each term of the L_1 distance function can be integrated separately, so

$$\mathrm{ED}(\vec{q},o_i) = \sum_{k=1}^{d} \mathrm{ED}_k(\vec{q},o_i) = \sum_{k=1}^{d} \int_{\mathbb{R}^d} w_k |q_k - x_k| f_i(\vec{x})\,\mathrm{d}\vec{x},$$

By marginalization, each term of the sum can be written

$$\mathrm{ED}_k(\vec{q},o_i) = w_k \int_{-\infty}^{\infty} |q_k - x_k| f_i(x_k)\,\mathrm{d}x_k.$$

Because $\vec{q}$ is outside the uncertainty region, for a particular dimension k, either (1) $q_k > x_k$ for all x_k such that $f_i(x_k) > 0$ or (2) $q_k < x_k$ for all x_k such that $f_i(x_k) > 0$. Assume without loss of generality that $q_k > x_k$. (The $q_k < x_k$ case is symmetric.) Then

$$\mathrm{ED}_k(\vec{q},o_i) = w_k q_k - w_k \int_{-\infty}^{\infty} x_k f_i(x_k)\,\mathrm{d}x_k = w_k(q_k - \mu_{ik}). \qquad \square$$

The fact that ED has such a simple shape outside the uncertainty region is promising: It means that an indexing scheme needs

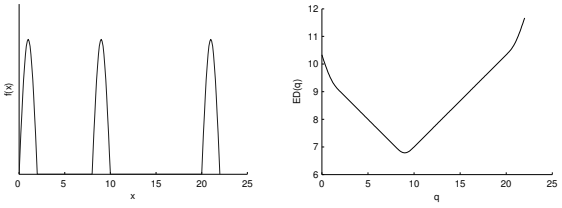


Figure 1. An example pdf (mixture of three Gaussians) and its ED-curve, which gives the expected distance from the pdf to any query point.

only concentrate on the part inside the uncertainty region. In the next section we will develop a compact approximation scheme that can be used to index ED under the L_1 distance—a distance measure that is a natural choice for joint distributions of attributes with different domains, such as temperature and pressure. Generalizing this to work with L_2 and other distance measures is future work.

Closed-form solutions can be found for ED for particular distributions—for instance, if f_i is uniform, then ED is parabolic inside the uncertainty region—but the focus of this paper is on arbitrary distributions.

4 Adaptive, piecewise-linear approximations

Sections 2 and 3 have established that probabilistic range and k-NN queries can be answered by estimating the value of a function at a certain point: For range queries the cdf of the object must be estimated at the extremes of the query range, and for k-NN queries the estimated distance (ED) must be estimated at the query point. We are now ready to present our approximation scheme, *Adaptive, Piecewise-Linear Approximations* (APLA).

APLA combines the virtues of the previous solutions discussed in Section 2.1: It (1) partitions the function adaptively; (2) approximates each partition by a linear function, which is likely a better fit than a constant function to fit the monotonically increasing cdf or the cup-shaped ED-curve; (3) it has a compact representation; and (4) index structures can be built for it.

We first describe the APLA representation and how upper and lower bounds for $F(x)$ can be computed for any x. We then present algorithms for computing an APLA, either directly from a set of observations or from other APLAs. APLA will be presented in terms of a one-dimensional function $F(x)$, then extended to two-dimensional functions (surfaces) in Section 4.2. Section 4.3 describes our APLA-tree index structure.

An APLA $\hat{F}(x)$ for a function $F(x)$ consists of a sequence of up to s line functions $L_1(x), L_2(x), \ldots, L_s(x)$, as well as a global error ε. The number of lines s is a parameter, and is the same for all objects in the database. Consecutive lines cannot be parallel, and a line's intersection with the next line of the approximation is required to have a larger x-coordinate than its intersection with the previous line.

For $i = 1, \ldots, s-1$, let $x_i^\cap$ be the x-coordinate of the point where L_i intersects L_{i+1}. Further, let $x_0^\cap$ and $x_s^\cap$ be the extreme points of the uncertainty region. (For cdfs, $x_0^\cap$ and $x_s^\cap$ can be found by solving $L_1(x_0) = 0$ and $L_s(x_s^\cap) = 1$, respectively; for ED-curves, the extreme points are stored with the APLA.) We call these $s+1$ points the *breakpoints* of $\hat{F}$. Given an APLA, the function can be estimated for any value of x as

$$\hat{F}(x) = \begin{cases} F(x) & \text{if } x \le x_0^\cap \\ L_1(x) & \text{if } x_0^\cap \le x \le x_1^\cap \\ L_2(x) & \text{if } x_1^\cap \le x \le x_2^\cap \\ \vdots & \\ L_s(x) & \text{if } x_{s-1}^\cap \le x \le x_s^\cap \\ F(x) & \text{if } x_s^\cap \le x. \end{cases}$$

The first and last case refer to the fact that outside the uncertainty region, the function need not be estimated, but can be computed exactly—for cdfs as either 0 or 1, and for ED-curves as the distance from the mean.

Let ε_i be the maximal absolute difference between a line L_i and the function F it approximates in the interval $[x_{i-1}^\cap, x_i^\cap]$, i.e.,

$$\varepsilon_i = \max_{x_{i-1}^\cap \le x \le x_i^\cap} |L_i(x) - F(x)|. \tag{3}$$

The global maximal error ε can then be defined as $\varepsilon = \max_{i=1}^{s} \varepsilon_i$. Thus, the APLA defines upper and lower bounds for every point x: We are guaranteed that $\hat{F}(x) - \varepsilon \le F(x) \le \hat{F}(x) + \varepsilon$.

Assuming that we already have partitioned the set of points sampled from $F(x)$ into subsets $X_1, X_2, \ldots, X_n$, how do we find the line $\hat{F}_i$ that approximates a subset X_i while minimizing the error ε_i? This problem can be expressed as a linear program as follows: Minimize ε subject to the linear constraints

$$z - \varepsilon \le a_i x + b_i \le z + \varepsilon \tag{4}$$

for each point $(x, z) \in X_i$. The problem can be solved quickly by computational geometry methods because there are only three variables (ε and the two line parameters a_i and b_i) [5].

There is one complication, however. Recall that, in the interest of compactness, the APLA contains only ε and the line parameters, so the breakpoints are used to determine the range in which line should be used as approximation. We therefore have to make sure that the breakpoints coincide with the partition boundaries. For reasons that will become clear in the proof of Theorem 1, we must not in fact partition the points, but find covering subsequences such that any two consecutive subsequences X_{i-1} and X_i overlap by one point x_k. We then augment the line fitting problem with constraints that ensure that the x-coordinate of the intersection of the lines $\hat{F}_{i-1}$ and $\hat{F}_i$ falls between $(x_{k-1} + x_k)/2$ and $(x_k + x_{k+1})/2$. When $a_{i-1} < a_i$, the applicable constraints are

$$\frac{x_{k-1} + x_k}{2}(a_i - a_{i-1}) \le b_{i-1} - b_i \qquad \text{and} \tag{5}$$

$$\frac{x_k + x_{k+1}}{2}(a_i - a_{i-1}) \ge b_{i-1} - b_i; \tag{6}$$

otherwise, they are

$$\frac{x_{k-1} + x_k}{2}(a_i - a_{i-1}) \ge b_{i-1} - b_i \qquad \text{and} \tag{7}$$

$$\frac{x_k + x_{k+1}}{2}(a_i - a_{i-1}) \le b_{i-1} - b_i. \tag{8}$$

When fitting a line, we first try constraints (5) and (6) with the additional constraint that $a_{i-1} < a_i$. We then fit the line again with constraints (7) and (8) with the additional constraint that $a_{i-1} > a_i$. The solution with the smallest ε_i is chosen.

Figure 2 shows the dynamic programming algorithm that adaptively computes an APLA for a function defined by N points. The

```
Algorithm COMPUTE-APLA
Inputs: P, a sequence of N points, sorted by x-coordinate
        s, the highest number of linear pieces to use
for ω = 1 to N − 1
   Fit a line L′ to points P[0] through P[ω]
   L[ω,0] := L′
   ε[ω,0] := the max. error of L′ (Eq. (3))
   if ω < N − 1 and P[ω].x = P[ω+1].x
      ε[ω,0] := ∞
   A[ω,0] := 0
for t = 1 to s − 1
   for ω = t + 1 to N − 1
      for α = t to ω − 1
         Fit a line L′ to points P[α] through P[ω]
         ε′ := the max. error of L′
         c := max{ε[α,t − 1], ε′}
         if ω < N − 1 and P[ω].x = P[ω+1].x
            ε[ω,0] := ∞
         if α > 0 and P[α − 1].x = P[α].x
            ε[ω,0] := ∞
         if α = t or c < ε[ω,t]
            L[ω,t] := L′
            ε[ω,t] := c
            A[ω,t] := α
t := s − 1
while t > 0 and ε[N − 1,t − 1] < ε[N − 1,t]
   F̂[t] := ∅
   t := t − 1
c := ε[N − 1,t]
ω = N − 1
while t > 0
   F̂[t] := L[ω,t]
   ω = A[ω,t]
   t := t − 1
return F̂[1]...F̂[s] and c
```

Figure 2. Dynamic programming algorithm for finding the adaptive, piecewise-linear approximation of a set of points

algorithm fills an $N \times s$ table ε such that $\varepsilon[\omega,t]$ is the max. error of the best t-segment approximation to points $0,\ldots,\omega$. Once the best $(t-1)$-segment approximation is known for every prefix of the points, the best t-segment approximation for points $0,\ldots,\omega$ can be found by computing

$$\varepsilon[\omega,t] = \min_{\alpha=t}^{\omega-1}(\varepsilon[\alpha,t-1]+\varepsilon_t), \tag{9}$$

where ε_t is the max. error of the line fitted to points α through ω. Previously fitted lines are kept in an array L so they can be used to compute constraints (5)–(8) during future line fittings. The array A keeps track of the best results: $A[w,t]$ is the index of the first point of the t-th segment of the best t-segment approximation to points $0,\ldots,\omega$. The algorithm ensures that if a line is fitted to points with a certain x-coordinate, it is fitted to all the points with that coordinate.

It is worth noting that constraints (5)–(8) can make the dynamic programming algorithm find a non-optimal APLA. To see why, consider two consecutive segments $\hat{F}_{i-1}$ and $\hat{F}_i$ where $\varepsilon_{i-1} < \varepsilon_i$. It is possible that a slightly suboptimal fit of $\hat{F}_{i-1}$ would allow $\hat{F}_i$ to fit its points better, reducing ε_i and thereby also the global error ε. Our experiments indicate that the algorithm still finds good approximations, however.

4.1 APLAs of other APLAs

Suppose that we have already computed APLAs $\hat{F}_1,\ldots,\hat{F}_n$ for functions $F_1,\ldots,F_n$ (cdfs or ED-curves). It is useful to be able to compute an APLA $\hat{G}$ for the entire set of functions without once again retrieving the large set of observations that define the functions. (This is particularly useful for building hierarchical index structures, such as the one described in Section 4.3.) The new APLA $\hat{G}$ can be computed as follows.

We begin by finding $X = \cup_{i=1}^{n}\{x : (x,z) \in X_i^{\cap}\}$, the set of all x-values that occur in breakpoints of any of the APLAs. For each $x \in X$, we compute the highest upper bound and the lowest lower bound over all the APLAs, i.e.,

$$z_{\max}(x) = \max_{\hat{F}_i}[\hat{F}_i(x)+\varepsilon_i] \qquad \text{and} \tag{10}$$

$$z_{\min}(x) = \min_{\hat{F}_i}[\hat{F}_i(x)-\varepsilon_i], \tag{11}$$

and add $(x, z_{\max}(x))$ and $(x, z_{\min}(x))$ to a set P. Finally, P is sorted by x-coordinate and passed to COMPUTE-APLA.

The following theorem proves that the computed APLA is indeed a valid approximation of the underlying distributions.

Theorem 1. *Let $\hat{F}_i$ be an APLA for function F_i. The APLA $\hat{G}$ computed from a set of APLAs that include $\hat{F}_i$ is a valid approximation for F_i, i.e., $\hat{G}$ satisfies the constraints of Ineq.* (4) *for all (x,z) s.t. $z = F_i(x)$.*

Proof. We say that an APLA $\hat{F}$ with max. error ε *satisfies* a point (x,z) if z is between the APLA's upper and lower bound at x, i.e., $\hat{F}(x) - \varepsilon \le z \le \hat{F}(x) + \varepsilon$. Suppose for the purpose of contradiction that (x,z) is a point that is satisfied by $\hat{F}_i$ but not by $\hat{G}$.

From X, the set of x-coordinates that occur in breakpoints, let a and b be the largest member less than x and the smallest member larger than x, respectively. (Note that a or b cannot be exactly x, for then $z_{\min}(x)$ and $z_{\max}(x)$ would be added to P. They span $\hat{F}_i(x) \pm \varepsilon_i$, so $\hat{G}$ would have been constrained to satisfy (x,z).) It follows that there cannot be any breakpoints with coordinates between a and b.

We say that a value x is *covered* by a line segment if the line was fitted to points with x-coordinates no greater than x and points with x-coordinates no less than x. (Recall that COMPUTE-APLA uses dynamic programming to find overlapping subsequences of x-coordinates from the input points, so most values are be covered by one line segment but some are covered by two.)

If a and b were covered by the same segment of $\hat{G}$, then $\hat{G}$ would satisfy (x,z). (The reason is that a and b are members of X, so the upper and lower bounds for $F(x)$ have been computed and added to P.) Consequently, there must be a segment of $\hat{G}$ that covers a and not b, and another that covers b but not a. Because COMPUTE-APLA fits lines to *overlapping* subsequences, there must then be a breakpoint with x-coordinate between a and b so that it can be covered by both segments. We already know that this is impossible, and this contradiction proves the theorem. □

The algorithms so far work well for computing APLAs for ED-curves and combining these APLAs. Cdfs, however, have two

properties that can lead to undesirable results when combining the APLAs of two or more cdfs. We address this in the following.

The first property is that the range of a cdf is limited to $[0,1]$. To see why this is a problem, consider the cdfs F_1 and F_2 of two univariate distributions with disjoint uncertainty regions. Assuming that F_1 is to the left of F_2, there a point x between the two uncertainty regions where $F_1(x) = 1$ and $F_2(x) = 0$. Any APLA that is is a valid approximation for both F_1 and F_2 will therefore have a maximal global error ε of at least 0.5. This makes the APLA useless, as the upper bound is 1 and the lower bound 0 for every x. The problem can be avoided by working with the inverse function of the cdf, thereby redefining ε_i to be the maximal absolute x-difference (instead of z-difference) between a line L_i and the function F it approximates. Special care must be taken if the cdf has constant intervals because the inverse is undefined in that case, but we omit the details.

The second property of cdfs that requires special attention is their monotonicity. Suppose an APLA $\hat{F}$ is computed for two objects with uniform pdfs. Object o_1's uncertainty region is $[1,2]$, and object o_2's uncertainty region is $[3,4]$. Thus, within the uncertainty regions, their cdfs are $F_1(x) = x-1$ and $F_2(x) = x-3$. With the constraints defined so far, the APLA computed for the objects is the N-shaped function

$$\hat{F}(x) = \begin{cases} x-1 & \text{if } 1 \le x \le 2, \\ -x+3 & \text{if } 2 \le x \le 3, \text{and} \\ x-3 & \text{if } 3 \le x \le 4, \end{cases}$$

which is certainly a good (perfect!) solution, but breaks the range search algorithm because the upper bound at $x = 2.5$ is lower than the lower bound at $x = 1.5$, so the computed probability that o_1 is in the range $[1.5, 2.5]$ is negative. The APLA can be forced to be monotone by constraining the line segments to have positive slope.

4.2 APLAs of joint probability distributions

The question of whether APLAs generalize to two or more dimensions is interesting from a theoretical standpoint. It is also of practical use when two or more quantities are observed together. For instance, the thickness of the cell layers can be measured at many locations on the retina, and the thickness of both the outer nuclear layer (ONL) and the inner nuclear layer (INL) are of interest. Biologists may want to ask for images that show an area of the retina where both the ONL and the INL are thick. Note that this is not the same as asking for images that show an area where the ONL is thick and an area where the INL is thick; the latter query is more permissive.

As a second example, consider a database of historical locations of taxicabs. Each location is a simultaneous observation of latitude and longitude. If a dispatcher wants to find all taxis that are in the northeastern part of town with probability at least 0.7, it is not enough to know that a certain car is likely to be on the east side and that it is likely to be on the northern part of town; we need to know that there is a high likelihood of the two observations occurring together.

The cdf of the joint distribution of two attributes a_1 and a_2 is a surface; the x and y axes are the domains of a_1 and a_2, respectively, and the z-axis is the cumulative probability: If the surface goes through a point (x,y,z), then the probability that $a_1 \le x$ and $a_2 \le y$ are observed together is z. The surface is xy-monotone, i.e., it is nondecreasing along both dimensions. A two-dimensional APLA is a compactly represented surface that approximates this surface, as well as a number ε indicating the greatest difference between the original surface and its approximation. Figure 3 shows an example of a bivariate cdf and its APLA.

Even ignoring the requirement that an APLA must be represented compactly, finding the optimal piecewise-linear approximation to a surface is a difficult problem. Agarwal and Suri [1] show that it is NP-hard to decide whether the surface can be ε-approximated with k triangles whose projections on the $z = 0$ plane are pairwise disjoint. We must therefore resort to heuristic algorithms.

A common heuristic for surface approximation is to maintain a triangulation of the surface, then greedily remove vertices whose incident planes are almost coplanar [1]. The resulting representation is not very compact, however: In addition to ε, we would have to store the x, y, and z coordinates of every remaining vertex. Instead, we select vertices so that their projections on the $z = 0$ plane form a grid. Thus, $p \times q$ vertices can be represented by p x-coordinates, q y-coordinates, and $p \times q$ z-coordinates.

The problem of selecting p x-coordinates and q y-coordinates is known as the $p \times q$ partitioning problem [15], and has been studied extensively because it comes up when mapping an irregular workload to nodes in a parallel computer. We adopt a simple heuristic approach that was independently proposed by Mingozzi et al. [13] and Nicol [16]. The algorithm starts with an arbitrary partitioning, then iteratively improves it. At each iteration, the partitioning along one axis is kept constant, and the optimal breakpoints are chosen along the other axis. The algorithm terminates when the partitioning no longer changes.

After finding a good partitioning, and thus the x and y-coordinates of the vertices, we are still one hurdle away from a compact APLA. As part of the process of finding a good partitioning, our algorithm fits planes to the partitions. (In fact, it fits two planes to each partition: one to the lower triangle and one to the upper.) Thus, each vertex has six incidental planes, each of which can have a different z-value at the vertex. How do we choose which z-value to store for each vertex? One could use the z-value from the original surface at the vertex, but this would lead to a large ε, as points in the middle of the partition can be far from the plane defined by its incidental points. Instead, we propose a heuristic algorithm that starts with the plane with highest ε. From the equation of the plane, we compute a new z-value for each of its three incidental vertices. This process is repeated for all the planes, in order of decreasing ε, assigning new values only to vertices that have not received one in previous steps.

The details of our algorithm has been omitted because of space constraints. We note that fitting a plane to a set of points is an instance of the L_∞-linear approximation problem, which can be solved efficiently by computing the convex hull (in $O(n \log n)$ time [5]) and finding the parallel planes of support with minimal separation (linear time in the size of the convex hull) [11].

Multidimensional APLAs are not the only way to answer a multidimensional query: One can also use a number of 1-D APLAs, one for each marginalized distribution $f(x_0), \ldots, f(x_n)$. An object is deemed to satisfy a range query if each of the one-dimensional APLAs satisfies the query. This is obviously more permissive than using a 2-D APLA: Just because there are observations of $x \in [x_0, x_1]$ and $y \in [y_0, y_1]$ does not necessarily mean that there are any observations of $(x,y) \in [x_0, x_1] \times [y_0, y_1]$, so using several 1-D APLAs may lead to lower precision. This effect is

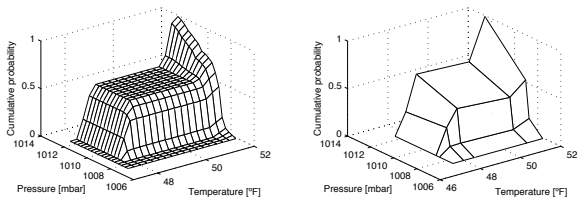


Figure 3. A bivariate cdf and its APLA.

counteracted, however, by the fact that $p \times q$ planes take up more space than $p+q$ line segments, so the space used by a p-by-q-plane 2-D APLA can fit two 1-D APLAs with more than p and q line segments, respectively.

4.3 Indexing the APLA

Because of its compactness, the APLA speeds up sequential scans. In order to reduce the time to answer queries further, we have designed a dynamic, balanced, paged index structure for the APLA. Our index structure, which we call the APLA-tree, is a tree structure similar to the well-known R-tree [7], but contains APLAs instead of rectangles. We only describe the differences.

Figure 4 illustrates the structure of the APLA-tree. A leaf node contains APLAs of one or more objects, as well as an object identifier. Probability distributions over d attributes are stored by a sequence of d APLAs, each approximating the marginalized distribution of one attribute. (Multidimensional APLAs, which approximate joint distributions of two or more attributes, can also be used.) An internal node consists of a number of entries. Each entry has a pointer to a child node and APLAs summarizing all objects in the subtree rooted at that child. If 1-D APLAs are used, there will be d APLAs in each entry (one for each dimension).

During range search, an upper bound for the appearance probability is computed for each entry in an internal node. Only if the appearance probability is at least τ is it necessary to retrieve the subtree rooted in the child node pointed to by the entry.

During k-NN search, two priority queues are maintained. There is a queue of the best objects seen so far, and also a second queue that contains nodes the subtrees of which have not yet been searched. The queue of nodes is sorted by the $\mathrm{ED}_{\min}$ of the subtree. The algorithm iteratively pops a node from the queue of nodes. Leaf nodes are processed as for sequential scan. For each entry of an internal node, $\mathrm{ED}_{\min}$ and $\mathrm{ED}_{\max}$ are computed. The child node pointed to by the entry is inserted in the queue of nodes only if $\mathrm{ED}_{\min}$ does not exceed the $\mathrm{ED}_{\max}$ of the k-th object in the queue of objects.

When inserting a new object with APLA $\hat{G}$ into the index structure, we recursively choose to insert the object into the subtree T_i with the lowest *cost*. (Ties are broken arbitrarily.) The cost $c(\hat{G},\hat{F})$ is the relative amount by which the ε of the APLA $\hat{F}$ of the subtree T_i would have to be increased in order to be a valid approximation for the new object. The cost can be computed as the maximal difference between $\hat{F}$ and any breakpoint of $\hat{G}$, adjusted by the global errors of the APLAs, i.e.,

$$c(\hat{G},\hat{F}) = \frac{\varepsilon_G - \varepsilon_F + \max\limits_{(x,z)\in X^{\cap}} |\hat{F}(x) - z|}{\varepsilon_F}, \qquad (12)$$

where ε_G and ε_F are the global errors of $\hat{G}$ and $\hat{F}$, respectively, and $X^{\cap}$ is the set of breakpoints for $\hat{G}$. If an entry contains more than one APLA (for different dimensions), $c(\hat{G},\hat{F})$ is computed for each APLA and the maximum used as the cost.

If inserting the new object would cause the node to overflow, the node is split, using a random split policy. Other split policies can be imagined using the same cost function as when choosing a subtree to insert into, but we leave a detailed investigation of the dynamic aspects of the index structure for future work. After splitting a node, new APLA are computed for the two resulting nodes, using the algorithm described in Section 4.1.

After inserting an object into a node, the APLA of the parent node may no longer be valid, so it must be adjusted or recomputed. To adjust an APLA means simply to increase its ε-value so as to satisfy the constraints Ineq. (4) for the new object. Recomputation is slightly more involved: Points are extracted from the APLAs of all children, and a new APLA computed for the parent using the algorithm described in Section 4.1. This paper does not focus on the dynamic aspects of the index structure, so we do not investigate the tradeoff of adjustment vs. recomputation.

5 Experimental results

This section evaluates the performance of APLA through experiments on synthetic and real data. All experiments were run on computers with Athlon MP 1800+ CPUs running Linux 2.6.8 and the LP-solver CPLEX 9.0.

We compared our technique to Tao et al.'s conservative functional boxes (CFBs) and the U-tree index structure based on them [19], and also to Korn et al.'s OptimalSplines [10]. A CFB, which is the approximation stored in the leaf nodes of a U-tree, consist of four lines, stored as eight numbers, for each dimension. For a fair comparison, we let our APLA technique use three line segments, which together with the global error take seven numbers to store. We padded each APLA with an eighth, unused number. We made OptimalSplines use three knots, which together with the global error also take seven numbers to store.

An object with a bivariate pdf was represented by two APLAs (one for each dimension, 128 bytes in total) and a 20-byte payload (object identifier, etc.). The page size was 4096 bytes, so 23 objects fit in each leaf node.

When constructing an APLA-tree for k-NN search, we used three-segment APLAs in the internal nodes, just as in the leaf nodes. For the APLA-tree for range search, however, we used two-segment APLAs in the leaf nodes. Again, this is so that the approximations in our internal nodes will use the same amount of space as the approximation in the U-tree's internal nodes, a requirement for fair precision experiments.

The first set of experiments performs range queries using both APLAs and CFBs, and compare the precision of the results. A low precision translates to high query time because there are many false positives, the bulky data of which must be retrieved from disk in a refinement step. For these experiments, we use the "LB" dataset, which was also used by Tao et al. [19] to evaluate the U-tree. The dataset consists of 53,144 two-dimensional constrained Gaussian distributions. The mean of each distribution is the central position of a geographical object in Long Beach, California, normalized to the $[0,10000]^2$ space. The standard deviations are 125, and the distributions are constrained to the area within two standard deviations from the mean. We varied the size of the query range from 500×500 to 2500×2500. For each size, we computed the average precision over 500 range queries with randomly placed

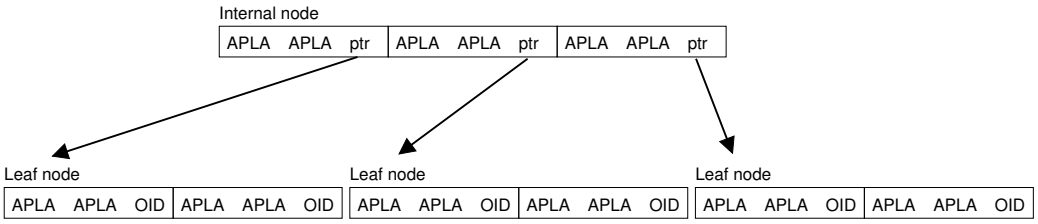
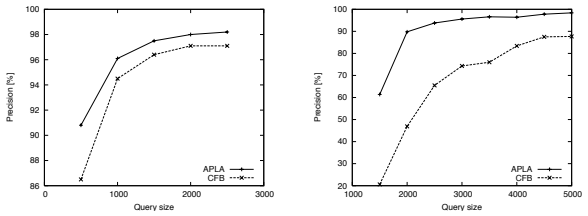


Figure 4. The structure of an APLA-tree. The APLAs in internal nodes approximate entire subtrees, and can be used for pruning.



(a) unimodal (b) bimodal

Figure 5. APLA yields only slightly better precision than CFB for the LB dataset because the CFB is already a good approximation for Gaussian distributions. For mixtures of two Gaussians, the precision of APLA is much higher than that of CFB.

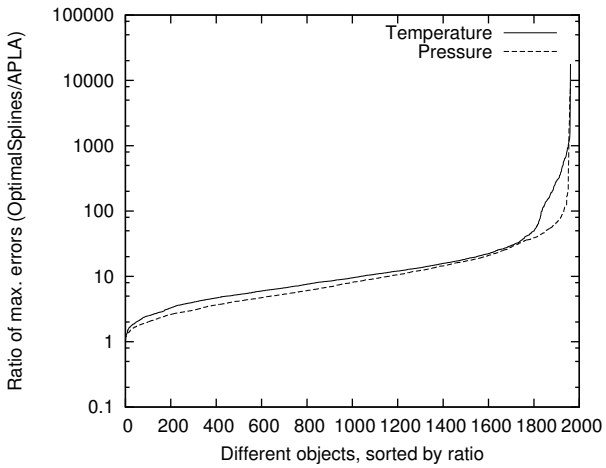


Figure 6. Because OptimalSplines minimizes the expected error, but the maximal error ends up being higher than APLA in all cases, and more than 10 times higher for about 50 % of the objects.

query ranges of the appropriate size and probability threshold 0.8. We answered the queries by sequential scan of the leaf nodes so as to isolate the performance of the object representations from the pruning performance of the tree structures.

We measured precision rather than running time in this experiment because the actual running time of a query would include refining the results in order to remove false positives. Because this experiment uses idealized distributions as ground truth, the time to refine the results would be very small, without any relation to the time it would take to query a set of arbitrary distributions, the refinement of which involves accessing actual observations on disk.

Figure 5(a) shows that although APLAs give higher precision than the CFBs for this Gaussian dataset, the improvements are modest. The reason is that the CFB is a pretty good envelope for a Gaussian cdf. In order to illustrate how the APLA can adapt to the distribution at hand and give decent approximations for arbitrary distributions, we created a new, bimodal dataset "LB-Bimodal" based on LB. In LB-Bimodal, each distribution is a mixture of two Gaussians, one 1250 units northeast of the other. The component Gaussians have standard deviation 125, as in LB. Figure 5(b) shows that the precisions of range queries on LB-Bimodal are dramatically different. For small queries, CFBs yield a precision as low as 20 %, and even for larger queries there is a 10–20 percentage point difference between the precisions of CFB and APLA.

The main conclusion to draw from the precision experiments is that a technique that approximates a Gaussian well does not necessarily approximate other distributions well, so a technique that targets arbitrary distributions must be evaluated using real datasets of empirical distributions. Consequently, we use a sensor network dataset for the remainder of our experiments. A network of 16 sensors [14] measured temperature and atmospheric pressure for $\sim$124 h. Each hour of measurements from each sensor was aggregated into one bivariate pdf, resulting in 1964 objects after dropping objects with very few samples. In order to get a larger dataset, we copied each object 35 times and shifted the copies to other locations in the data space.

We computed APLAs and OptimalSplines for each of the objects and compared the global error. Figure 6 shows that OptimalSplines had larger error in every case, and for about half the objects the error was more than ten times that of APLA. The reason is that OptimalSplines minimizes the expected error instead of the maximal error, so the error can be high in small parts of the domain. The high maximal error destroys the pruning power for all queries.

5.1 Range query performance

In this section, we compare the range query performance of the APLA-tree to that of the U-tree using the sensor network dataset. The index structure is about one-twentieth of the size of the actual data, so it is reasonable to assume that the index structure can be cached in main memory and the actual data cannot. Before each query, the file system was therefore unmounted and remounted in order to flush the operating system's cache. The index structure was then scanned sequentially so as to bring it into the cache.

We measured the total time to answer a query for all objects that have temperatures between 55 and 58 degrees and pressure

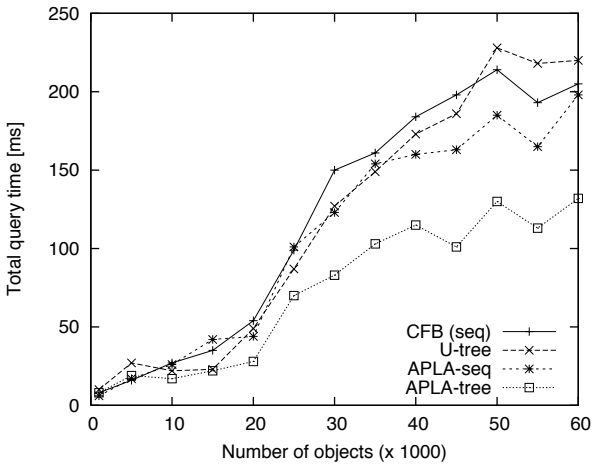


Figure 7. For range queries, the U-tree yields no noticeable performance improvement over sequential scan on CFBs because the CFBs' precision on this dataset is too low. Sequential scan of APLAs is 15 % faster, and the APLA-tree is twice as fast.

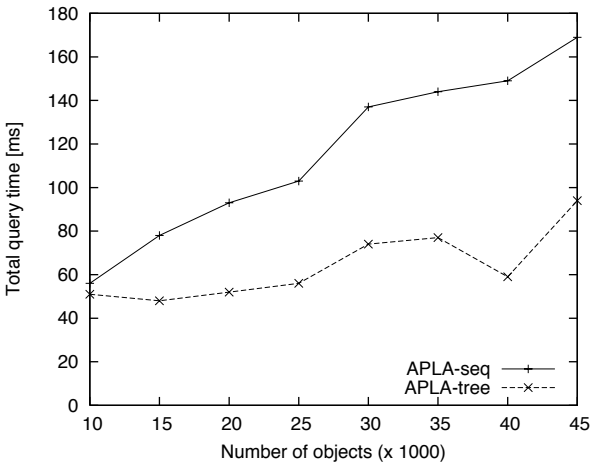


Figure 8. Total time to answer a 10-NN query. APLA-sequential is linear, but 50 times faster than the naïve approach (not shown), and APLA-tree is twice as fast as APLA-sequential.

between 1005 and 1010 mbar with a probability of at least 0.9. The time includes a refinement step to ensure that there are no false positives among the results.

Figure 7 plots the total time to answer range queries as a function of database size for Tao et al.'s techniques (CFB and U-tree) as well as our APLA-seq and APLA-tree. The graph shows that for the sensor dataset, the U-tree is not much faster than sequential scan of CFBs. The reason is that the precision of CFB is lower than that of APLA (42 % compared to 67 %), so the U-tree is not able to prune enough subtrees to gain an advantage. The sequential scan using APLAs is about 15 % faster, and the APLA-tree is about twice as fast as the other techniques.

5.2 k-NN query performance

APLAs of the ED-curve can be used to answer k-NN queries both by sequential scan and using the APLA-tree structure. Because the APLAs yield a range for the expected distance to each object, the searches do not return exactly k objects, but a larger set that are guaranteed to contain the k nearest neighbors. This set must then be *refined* by retrieving the objects from disk and computing their actual expected distances to the query.

Figure 8 shows that a sequential scan of APLAs can answer a 10-NN query on 30,000 objects in 144 ms. The APLA-tree is a factor of two faster, and answers the same query in 74 ms. The times plotted are the total times to search the index structure, then refine the results by retrieving actual objects from disk.

Because this paper is the first to answer k-NN queries on arbitrary probability distributions, there are no competing techniques to compare to, but we note that computing the expected distance to each object directly by reading all observations from disk, is far too slow to be practical: The time increased linearly from 2.8 s for 10,000 objects to 8.7 s for 45,000 objects. (The observations were stored in binary format so no parsing was necessary.)

In order to better understand where the time to answer k-NN queries is spent, Figure 9 plots the CPU and I/O time for sequential

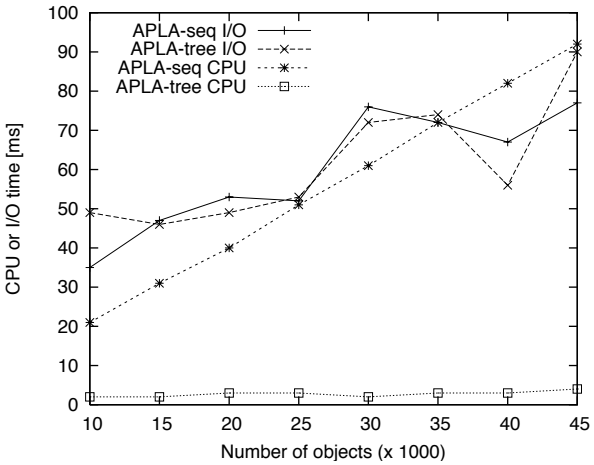


Figure 9. CPU and I/O time for 10-NN queries on datasets from 10,000 to 45,000 objects. The CPU time of sequential scan increases linearly, and becomes very significant, but the CPU time of the tree-based search stays below 4 ms.

and tree-based 10-NN search. The CPU time was obtained by not refining the results from the search, and the I/O time is the difference between total time and CPU time. The CPU time for sequential scan increases linearly, and at around 30,000 objects it starts to surpass the time required to refine the results. In contrast, the CPU time for the tree-based search remains below 3 ms.

We note that both APLA-seq and the APLA-tree answer 10-NN queries with more than 80 % precision (in most cases more than 90 %). As a consequence, the number of candidate objects that need to be refined are only 11 or 12 in most cases. For many applications, this is sufficiently precise, so the refinement step can be skipped, and because the index structure fits in the cache, a 10-NN query on 40,000 objects answered in 4 ms.

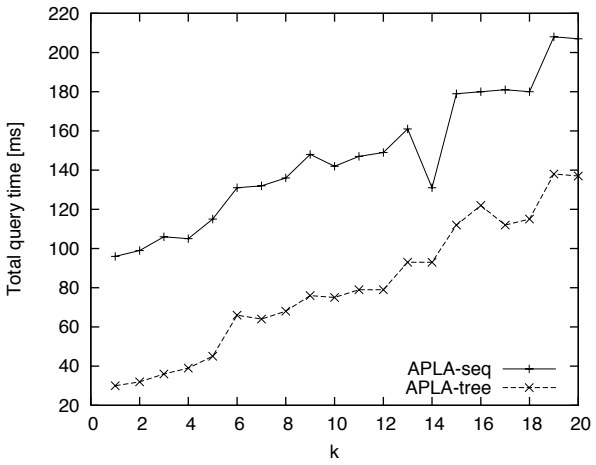


Figure 10. On the 30,000-object dataset, the APLA-tree answers k-NN queries twice as fast as APLA-sequential, regardless of k.

Figure 10 plots total query time as a function of k for the 35,000-object dataset. We see that the query time increases linearly with k. The cost of searching the index structure remains constant at 72 ms for APLA-sequential and 3 ms for APLA-tree, so the increase is because the number of candidates that must be refined goes up. The APLA-tree is twice as fast as APLA-sequential, regardless of k.

6 Conclusion

Querying databases of uncertain values is important for many applications, ranging from moving object databases to sensor networks and biomedical and other scientific databases. Probabilistic databases must handle arbitrary, empirical probability distributions in order to support data with uncertainty models that are complex or poorly understood.

We have presented algorithms for computing adaptive, piecewise-linear approximations (APLAs) of arbitrary functions. We have applied the APLA technique to cumulative distribution functions and shown that APLAs provide more precise approximations than existing methods, and therefore answers queries faster. We have proposed to answer k-NN queries by ranking objects by their expected distance to the query point, and applied the APLA technique to the expected distance (ED) function. Finally, we have described a dynamic, balanced, paged index structure for APLAs and shown experimentally that it speeds up both range and k-NN queries by a factor of two.

In the future, we will apply APLA to biomedical and other datasets in order to understand the datasets and queries for which 2-D APLAs are advantageous. We also plan to improve the heuristics for finding two-dimensional APLAs that are both compact and precise. Finally, we will investigate and find ways to index expected distance (ED) functions for L_2 and other distance measures.

Acknowledgements

The sensor network data used for the experiments were collected by Greg Moore from RACELab, the laboratory of Chandra Krintz, at UCSB [14]. The LB dataset was provided by Yufei Tao [18]. This work was supported in part by grants no. ITR-0331697 and EIA-0080134 from the National Science Foundation.

References

[1] P. K. Agarwal and S. Suri. Surface approximation and geometric partitions. *SIAM J. Comput.*, 27(4):1016–1035, Aug. 1998.

[2] C. Böhm, A. Pryakhin, and M. Schubert. The Gauss-tree: Efficient object identification in databases of probabilistic feature vectors. In *Proc. ICDE*, 2006.

[3] R. Cheng, D. V. Kalashnikov, and S. Prabhakar. Querying imprecise data in moving object environments. *IEEE Transactions on Knowledge and Data Engineering*, 16(9):1112–1127, Sept. 2004.

[4] R. Cheng, Y. Xia, S. Prabhakar, R. Shah, and J. S. Vitter. Efficient indexing methods for probabilistic threshold queries over uncertain data. In *Proc. VLDB*, 2004.

[5] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry*. Springer, 2000.

[6] A. Faradjian, J. Gehrke, and P. Bonnet. GADT: A probability space ADT for representing and querying the physical world. In *Proc. ICDE*, pages 201–211, 2002.

[7] A. Guttman. R-trees: A dynamic index structure for spatial searching. In *Proc. SIGMOD*, pages 47–57, 1984.

[8] H. V. Jagadish, N. Koudas, S. Muthukrishnan, V. Poosala, K. C. Sevcik, and T. Suel. Optimal histograms with quality guarantees. In *Proc. VLDB*, pages 275–286, 1998.

[9] E. Keogh, K. Chakrabarti, S. Mehrotra, and M. Pazzani. Locally adaptive dimensionality reduction for indexing large time series databases. In *Proc. SIGMOD*, pages 151–162, 2001.

[10] F. Korn, T. Johnson, and H. Jagadish. Range selectivity estimation for continuous attributes. In *SSDBM*, pages 244–253, 1999.

[11] D. Lee and Y. Wu. Geometric complexity of some location problems. *Algorithmica*, 1(2):193–211, 1986.

[12] M. Martone, S. Zhang, A. Gupta, X. Qian, H. He, D. Price, M. W. M, S. Santini, and M. Ellisman. The Cell-Centered Database: A database for multiscale structural and protein localization data from light and electron microscopy. *Neuroinformatics*, 1(3):379–396, 2003.

[13] A. Mingozzi, S. Ricciardelli, and M. Spadoni. Partitioning a matrix to minimize the maximum cost. *Discrete Applied Mathematics*, 62(1–3):221–248, 1995.

[14] G. Moore. RACELab sensor network project. `http://www.cs.ucsb.edu/~ckrintz/racelab/sensornet/sensornet.html`, Oct. 2005.

[15] S. Muthukrishnan and T. Suel. Approximation algorithms for array partitioning problems. *J. Algorithms*, 54(1):85–104, Jan. 2005.

[16] D. Nicol. Rectilinear partitioning of irregular data parallel computations. *J. Parallel Distr. Comp.*, 23(2):119–261, Nov. 1994.

[17] A. K. Singh, B. S. Manjunath, and R. F. Murphy. A distributed database for bio-molecular images. *SIGMOD Record*, 33(2):65–71, 2004.

[18] Y. Tao. A dataset collection. `http://www.cs.cityu.edu.hk/~taoyf/ds.html`, Oct. 2005.

[19] Y. Tao, R. Cheng, X. X. W. K. Ngai, B. Kao, and S. Prabhakar. Indexing multi-dimensional uncertain data with arbitrary probability density functions. In *Proc. VLDB*, 2005.