Adaptive Density Estimation

Arturas Mazeika
Free University of Bozen-Bolzano
Dominikanerplatz-3
Bozen-Bolzano, Italy
arturas@inf.unibz.it

Michael H. Böhlen
Free University of Bozen-Bolzano
Dominikanerplatz-3
Bozen-Bolzano, Italy
boehlen@inf.unibz.it

Andrej Taliun
Free University of Bozen-Bolzano
Dominikanerplatz-3
Bozen-Bolzano, Italy
taliun@inf.unibz.it

ABSTRACT

This demonstration illustrates the APDF tree: an adaptive tree that supports the effective and efficient computation of continuous density information. The APDF tree allocates more partition points in non-linear areas of the density function and fewer points in linear areas of the density function. This yields not only a bounded, but a tight control of the error. The demonstration explains the core steps of the computation of the APDF tree (split, kernel additions, tree optimization, kernel additions, unsplit) and demos the implementation for different datasets.

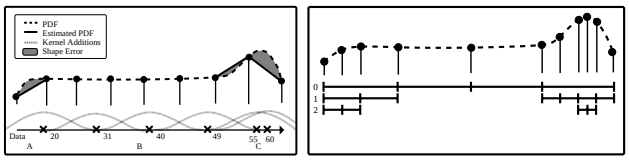
1. INTRODUCTION

Density is common statistical information that is used for approximate query answering, query optimization, clustering, etc. Typically, histogram [1, 3] are used to roughly estimate the density. Kernel based estimators generalize histograms [4]. They ensure that the estimated density function is continuous and that derivatives exist. This demonstration demos the APDF tree [2], an adaptive tree that supports the effective and efficient computation of continuous density information.

Figure 1(a) illustrates kernel additions. The data points are illustrated by the six crosses at the bottom of Figure 1(a). We draw a kernel function around each data point. The addition of all kernels yields the estimation of the probability density function (PDF, dashed line).

The kernel additions are computed on a uniform partition, and the values of the PDF are interpolated between partition points. The shaded regions quantify the *shape error* (SE): the difference between the PDF and the linear interpolations. The figure illustrates the key problem with a uniform partitioning: a fine partitioning yields a good but slow estimation whereas a coarse partitioning yields a bad but fast estimation. The ADPF tree is an adaptive tree structure that yields a fast and precise estimation of the PDF.

VLDB '06, September 12-15, 2006, Seoul, Korea.



(a) Uniform Partition (b) The APDF Tree

Figure 1: The Idea of the APDF Tree (1D)

Figure 1(b) illustrates the APDF tree for the same dataset. The APDF tree allocates more partition points in the non-linearity areas of the PDF and fewer points in the linearity areas of the PDF. This yields not only a bounded, but a tight control of the shape error. This makes the estimation fast and precise. The paper discusses the computation of the APDF partitions with a tight control of shape error and demos the implementation for different data.

We organize the demonstration in five main parts:

(i) Overview of the APDF method. The construction of the partition is an iterative process. We start with a uniform partition and show how the partition is refined in areas with a high shape error.
Goal: Introduce the elemental concepts of the APDF method: *smallest partition*, *directional splits* and *areas of high shape error*.
Demoed datasets: PlaneSphere (2D).

(ii) Steps of a single iteration of the APDF method. We zoom into one iteration of the method and demo the individual steps of the iteration.
Goal: Introduce the core building blocks of the APDF method: Split, TreeOptimization I, Kernel Additions, UnSplit, TreeOptimization II. Demo of these steps.
Demoed datasets: PlaneSphere (2D)

(iii) Evaluation of the APDF method for different datasets. We show how the APDF partition reflects the subdimensionality of the structures and the areas of non-linearity in the database.
Goal: Compare the final partitions of the APDF tree for different datasets, and different dimensionality.
Demoed datasets: PlaneSphere data, Linear dataset, Clickstream data, Plane (2D plane in 3D) PlaneHole

(2D plane with a hole in 3D), Spiral (spiral in 3D), Cone (3D).

(iv) Visual comparison of the APDF partition with histogram partitions: equi-width, wavelet based, GENHIST, etc.
Goal: Compare the partition of the APDF estimator wrt state-of-the-art histogram partitions.
Demoed datasets: PlaneSphere (2D).

(v) Robustness of the APDF tree. We select extreme datasets and parameters to illustrate the complexity of the problem and give a feel for the application area of the APDF tree.
Goal: Investigation of the estimator for very large datasets, very high precisions, and very sparse initial grid.
Demoed datasets: PlaneSphere (2D).

The paper is organized as follows. The overview part of the demo is presented in Section 2. Section 3 describes the steps of one iteration. The evaluation of the APDF tree is presented in Section 4. Finally, we sketch the algorithm for the construction of the APDF tree in Section 5.

2. OVERVIEW OF THE APDF METHOD

We use the PlaneSphere dataset (cf. Figure 2) to illustrate the construction of the APDF tree. The dataset consists of two structures: the plane (distributed in the unit square, and a sphere (2D normal distributed data in the bottom-right corner of the unit square).

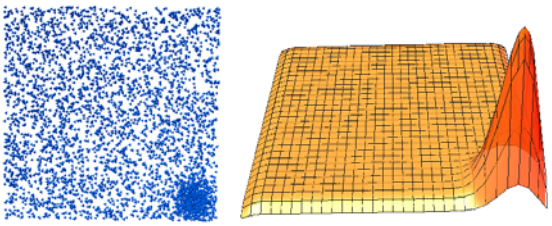


Figure 2: The PlaneSphere Data and its PDF

Figure 3 illustrates the creation of the APDF tree for the PlaneSphere dataset. The computation starts with a sparse uniform partition and kernel additions on this partition (cf. Figure 3(a). The APDF method estimates areas of non linearity of PDF and splits the corresponding areas. In our example all areas are split after the first iteration (cf. Figure 3(b)). This process continues until the approximated shape error is uniformly low in all areas of the APDF tree. In iteration 2 (cf. Figure 3(c)) only the areas at the borders are split. Since the PDF is linear in the center of the space, no additional points are introduced there. The 3rd iteration (cf. Figure 3(c)) adds further partition points at the borders of the universe. The rectangles are split in X and Y direction in the corners of the universe. The PDF is non-linear in both directions there. The rest of the rectangles are split in one direction since the PDF is non-linear only in one direction. The 4th iteration completes the creation of the APDF tree. Only the area with the sphere is split. The PDF is non-linear at the peak point of the PDF and at the boundary of the sphere.

3. INDIVIDUAL STEPS

This section zooms into the 3rd iteration and presents the core steps of one iteration of the APDF tree construction: *split*, *tree optimization*, *kernel additions*, *unsplit*, and the second tree optimization.

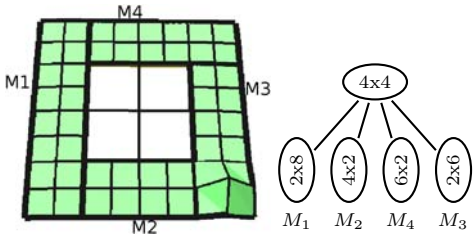


Figure 4: Start of the Iteration (cf. Figure 3(c))

Figure 4 shows the APDF tree at the beginning of the iteration. The tree consists of the root node with a uniform partition of size 4x4 (4 partition points per coordinate). 12 outer rectangles of the root are split, and 4 inner rectangles are not split. The 12 split rectangles are organized into 4 (local uniform partition) nodes of sizes 2x8, 4x2, 6x2, and 2x6. The tree illustration shows the granularity of the nodes and the parent-child connections. The areas that were split in the previous iteration are colored green. These are the only areas that can have a too high shape error.

The split step is the first step in the iteration. The split step processes the set of nodes C that were introduced in the previous step. It scans C and estimates whether the shape error is too high along one of the coordinates. If the shape error is too high, it splits according to the coordinate. Since the split predicts the shape error it can (and often will) introduce too many splits. The unsplit step is later used to remove unnecessary splits.

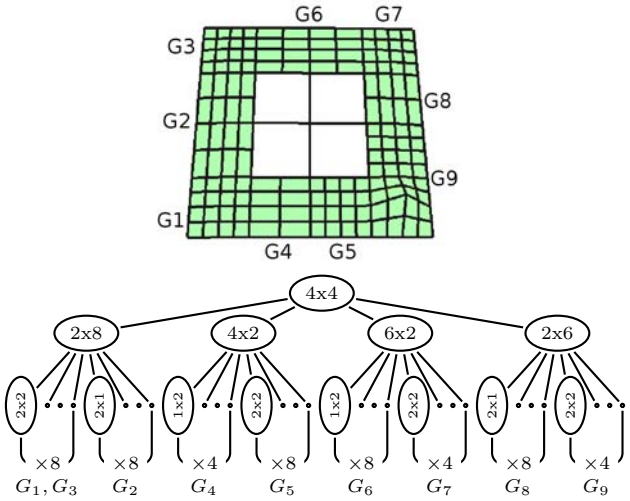
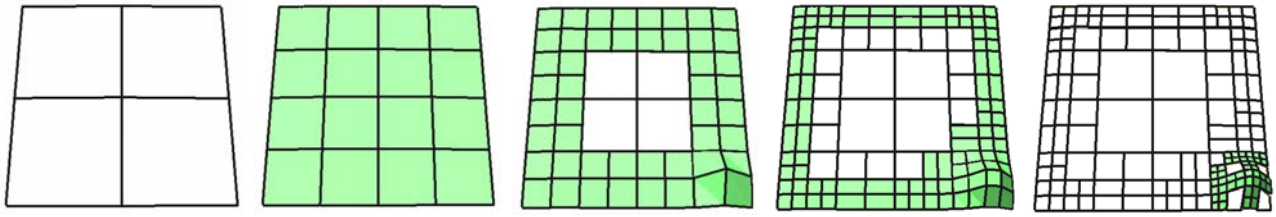


Figure 5: Split

Figure 5 illustrates the result of the split step. All areas were split further. In the corners of the unit square the PDF is non-linear in both dimensions, therefore we split the areas according to both dimensions. In the other areas the PDF



(a) Initial Partition (b) 1st Iteration (c) 2nd Iteration (d) 3rd Iteration (e) 4th Iteration

Figure 3: Creation of the APDF Tree

is non-linear according to one dimension only, therefore the rectangles are split according to one dimension only. The split step produces a lot of small rectangles (the group of rectangles is indicated by G_i). We use the three optimization step to group the rectangles into a fewer nodes.

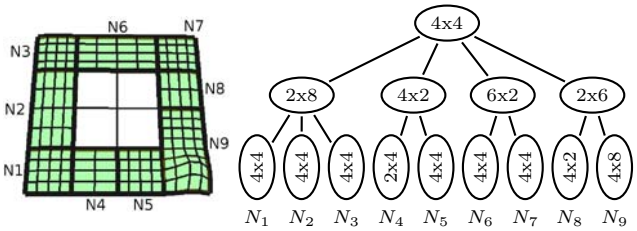
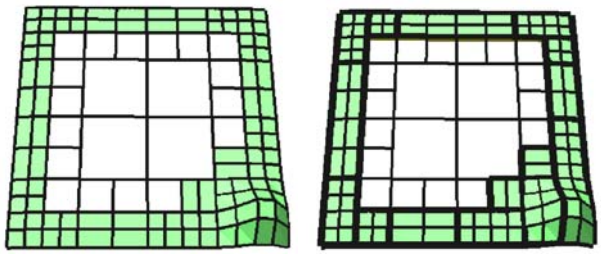


Figure 6: Tree Optimization

The tree optimization (TO) step is applied to the APDF tree to speed up the computation of kernel additions. The TO step scans the nodes C that were split and groups nodes of the same granularity into new nodes of *local uniform partitions* (LUP). The algorithm produces hyper-rectangular shaped nodes (cf. Section 5). Figure 6 shows the APDF tree after the tree optimization. The TO step reorganizes the 52 nodes introduced by the split step into 9 nodes. This effectively reduces the time spent during the subsequent kernel additions. The kernel addition step updates the new partition points with kernel additions.



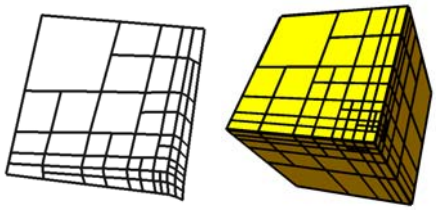
(a) Unsplit (b) Tree Optimization (TO)

Figure 7: Unsplit and Tree Optimization Steps

The unsplit (US) step scans the nodes and removes partition points that did not increase the precision of the estimator. At these points the shape error was already low and the split step over-split the area. The partition after the US step is illustrated in Figure 7(a). The second tree optimization completes the iteration (cf. Figure 7(b)). Again contiguous areas of the same shape are grouped into the same node to get large areas with a local uniform partition.

4. EVALUATION OF THE APDF TREE

This section describes the third part of the demo: the comparison of the APDF partitions for different datasets and dimensions. Figure 8 illustrates the APDF trees for the Linear dataset. The distribution of the data is linear according to each coordinate. The density increases from corner $(0, 0, 0)$ towards the opposite corner of the universe where it decreases very fast.

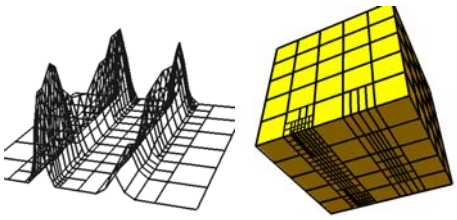


(a) 2D APDF (b) 3D APDF

Figure 8: The Linear Dataset

The dataset illustrates the directional splits of the APDF tree very nicely. The APDF tree does not allocate any partition points in the area of linearity of PDF, and introduces directional splits in the areas of non-linearity of the PDF.

Figure 9 illustrates the APDF tree for real world click stream data. Country, time of visit, and URL of the retrieved document are mapped to X, Y, and Z coordinates. The PDF reflects that most clicks come from two countries and that they have a different distribution along the time dimension. The APDF tree introduces directional splits in the areas of the non-linearity of the PDF, and does not split the rest of the universe.



(a) 2D APDF (b) 3D APDF

Figure 9: The Click-Stream Dataset

5. ALGORITHMS

Figure 10 presents the algorithm for constructing the APDF tree. The algorithm consists of two parts: the initialization part (lines 1–4) and the loop over the five core steps (line 5). The initialization part initializes the data structure with uniform partition, computes kernel estimation parameters, and augments kernel additions on the initial partition. The loop iterates the five steps of the APDF method until the shape error is lower than the given precision ε in all areas of space.

```
Algorithm: Construct_APDF_Tree

Input:
  Database: X[i] = (X1[i], X2[i], X3[i]), i = 1, ..., n
  Precision ε
  Initial partition P^I or the granularity g of
    the input uniform partition
Output:
  APDF tree a
Body:
  1. Initialize a with the given initial partition P^I or
     a uniform partition of granularity g. The tree consists
     of the root node only.
  2. Calculate estimation parameters: mean, variance, and h_opt
  3. Augment Kernels on the root node:
     KernelAdditions(root, X[i]).
  4. SE=∞; nodes_high_SE = root;
  5. WHILE nodes_high_SE ≠ ∅ DO
       5.1 new_leaves= Split(nodes_high_SE)
       5.2 opt_leaves= TreeOptimization(new_leaves)
       5.3 KernelAdditions(opt_leaves, X[i])
       5.4 nodes_high_SE= UnSplit(opt_leaves)
       5.5 nodes_high_SE= TreeOptimization(nodes_high_SE)
```

Figure 10: Construct APDF Tree Algorithm

Below we discuss the five core steps of the APDF algorithm in more detail.

The `Split` step processes all candidate nodes and splits the hyper-rectangles with a high shape error. Conceptually, the check for high shape error in a cube is done in the following way. Let P_L, P_M, and P_R be three adjacent partition points. Let $LI_{P_L,P_R}(P_M)$ be the linear interpolation value between P_L and P_R at point P_M. If

$$|f(P_M) - LI_{P_L,P_R}(P_M)| > \varepsilon \quad (1)$$

then the shape error around line P_L, P_R is too high. This idea is implemented in the following way. We scan the input nodes; for each input node we scan the dimensions; for each dimension we scan all adjacent triplets that are on the line parallel to the dimension. If Equation (1) is satisfied for the triplet, we split all hyper-rectangles surrounding the line.

The `Split` step introduces individual hyper-rectangles of a very small granularity. The `TreeOptimization` step groups individual hyper-rectangles of the same granularity into a single hyper-rectangle. This step is implemented in the following way. While the set of individual hyper-rectangles is not empty the algorithm takes a random hyper-rectangle C from this set and tries to add individual hyper-rectangles around C in all dimensions. If the expansion in all dimensions is impossible we try to expand in all dimensions but one. When that becomes impossible we try to expand in all dimensions but two. We continue this process until it is impossible to expand in any dimensions. This reorganizes individual hyper-rectangles into large hyper-rectangles.

The `KernelAdditions` step augments kernel additions on newly introduced nodes. The step is implemented in the following way. For each data point it scans the new nodes. For each new node it augments the kernels on the partition points in the node. The efficiency of the kernel additions increases, as the number of individual new nodes decreases.

The `UnSplit` step removes the splits in the areas where the `Split` step over-split the hyper-rectangles. The step scans the new nodes: for each dimension it checks adjacent triplets of partition points and determines if the middle point can be removed without loss of precision. If so the point is marked for removal. At the end of the UnSplit step all marked partition points are removed, which results in hyper-rectangles with a small cardinality. These hyper-rectangles are grouped with the second tree optimization step.

6. SUMMARY

The demonstration discusses the efficient estimation of continuous density information, which is important statistical information that is used in many areas (e.g., approximate query answering, clustering, query optimization). Our estimator not only provides an upper but also a lower bound for the error. This allows to control the space and time complexity of the estimation. A variety of data sets will be available that allow to investigate the estimation of density information in general and the construction of the APDF tree in particular.

7. REFERENCES

[1] A. Aboulnaga and S. Chaudhuri. Self-tuning histograms: building histograms without looking at data. In *SIGMOD*, 1999.

[2] APDF-DS Method. http://www.inf.unibz.it/dis/projects/3dvdm/. [Online; accessed 09-June-2006].

[3] A. Deshpande, M. Garofalakis, and R. Rastogi. Independence is good: dependency-based histogram synopses for high-dimensional data. In *ACM SIGMOD*, 2001.

[4] B. Silverman. *Density Estimation for Statistics and Data Analysis*. Chapman & Hall, London, 1986.