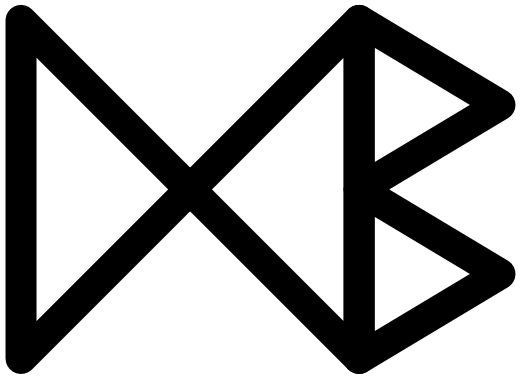


Parameter Estimation for Interactive Visualisation of Scientific Data

Albrecht Schmidt, Michael H. Böhlen

February 10, 2004

TR-4

A DB Technical Report

Title	Parameter Estimation for Interactive Visualisation of Scientific Data
Author(s)	Albrecht Schmidt, Michael H. Böhlen
Publication History	February 2004. A DB Technical Report.

For additional information, see the DB TECH REPORTS homepage: ⟨www.cs.auc.dk/DBTR⟩.

The DB TECH REPORTS icon is made from two letters in an early version of the Rune alphabet, which was used by the Vikings, among others. Runes have angular shapes and lack horizontal lines because the primary storage medium was wood, although they may also be found on jewelry, tools, and weapons. Runes were perceived as having magic, hidden powers. The first letter in the logo is "Dagaz," the rune for day or daylight and the phonetic equivalent of "d." Its meanings include happiness, activity, and satisfaction. The second letter is "Berkano," which is associated with the birch tree. Its divinatory meanings include health, new beginnings, growth, plenty, and clearance. It is associated with Idun, goddess of Spring, and with fertility. It is the phonetic equivalent of "b."

Abstract

This paper presents a method for accelerating algorithms for computing common statistical operations like parameter estimation or sampling on B-Tree indexed data in the context of visualisation of large scientific data sets. The method builds heavily on technology that is already part of relation database management systems and requires only small extensions. The technical goal is the following: Given a massive set of scientific data like sensor data stored in a Relational Database Management System, enable interactive exploration and visualisation of the data by exploiting the technology that already is in place in the database back-end. The main underlying idea is the following: the shape of balanced data structures like B-Trees encodes and reflects data semantics according to the balance criterion. For example, clusters in the index attribute are somewhat likely to be present not only on the data or leaf level of the tree but should propagate up into the interior levels. The paper investigates opportunities and limitations of this approach for visualisation. The advantages of the method are manifold. Not only does it enable advanced algorithms through a performance boost for basic operations, but it also builds on functionality that is already present to a large degree in current RDBMSs; furthermore, it is fully dynamic by avoiding redundancy: when the underlying source data change, the index and therefore the estimations adapt accordingly. Furthermore, we show that the sample quality is data-independent and never worse than a uniform sample if some basic prerequisites are ensured.

1 Introduction

There is considerable interest in using database management technology for scientific applications [36]. Deploying Relational Database Management Systems (RDBMSs) in these settings equips database users with declarative query languages and database administrators with a plethora of tools useful for managing large amounts of data. In line with researchers who seek to draw benefit from technology that was originally designed to support managing administrative and business data, the authors of this paper show how to exploit the most common database indexing method, namely the B-Tree [5], for accelerating visualisation algorithms [21] in scientific applications.

In more detail, the application setting of our research is the following: Given a set of scientific data such as raw sensor data or already pre-processed experimental data that are stored in a Relational Database Management System (RDBMS), estimate properties of the data and render views of the data that are later useful for visualising certain aspects of the data. Since this kind of application is often *exploratory*, *i.e.*, users are mostly interested in identifying regions of interest in the data, exact answers, which tend to be resource-consuming, are not required. Rather, an approximate but early answer is usually not only sufficient but desirable and allows more and immediate user interaction. Approximate answers are even more useful when there exist error estimates or error guarantees for a query [16].

Databases in scientific processing tend to comprise massive data volumes that are often measured in peta bytes [37]. At the same time, it is desirable to constrain users in the choice of queries as little as possible and allow for querying according to a variety of criteria. Therefore, tools known from other approaches to handling massive data like Data Warehousing [19] tend not to work in scientific settings since their optimisation techniques like indexed views or pre-aggregation are very hard to adapt to new types of queries as they are designed with a very specific set of queries in mind. This holds even more if exact results are required. Additionally, trial queries and previews play a major role in interactive settings [16]; once a region of interest has been identified users can refine their queries now against a much smaller part of the database and thus use computationally more demanding algorithms. The algorithms presented in this paper are designed to enable such an interactive querying paradigm. Our starting point is a sample of the original source data set extracted from a B-Tree index.

For the better part of this article, we concentrate on the two topics of parameter estimation [33] and sampling techniques which are an integral part of many visualisation algorithms. Broadly speaking, obtaining good estimates of, *e.g.*, densities, and samples enables advanced visualisation applications in a number

of ways. By being provided with the means for identifying regions of interest, users are equipped with the tools necessary to determine starting points for browsing the data set. The output of an initial approximate analysis can be the starting point for multi-resolution analyses, such as estimation of fractal dimensionalities for further data analysis, or noise reduction, contour shells, contour surfaces [30], which would be too resource-intensive to be executed on large datasets.

More formally, our application scenario is the following: Given a set D of tuples D_i with a key K_i which may be part of the data or a surrogate, we store the pairs $\langle K_i, D_i \rangle$ in a B-Tree using the K_i as keys and the D_i as data. The main underlying assumption for all ideas presented in this paper is that the data found on upper levels of a B-Tree, *i.e.*, a subset of the K_i, can be used as a sample that is representative for the complete data set. We will later show experimentally, and for special cases also analytically, that the sample is indeed well-behaved in our application domain and that, especially on large data sets, this sample can be obtained much faster than conventional methods which involve linear scans. With respect to the well-behavedness of the sample we can put forward a probabilistic argument: the re-balancing algorithm of the B-Tree is likely to provide good sampling (without replacement) with respect to upper levels of the tree. Indeed, an investigation of bulk loading algorithms for B-Trees reveals that a very regular sample is produced, *i.e.*, every k-th K_i in the ordered $\langle K_i, D_i \rangle$ is part of the sample (for some fixed k). For trees that are the result of random insertions, our experiments indicate that independent of insertion order and independent of the data properties the sample behaves like a uniform sample.

Depending on the actual application domain, different interpretations of the sample obtained are sensible. One could regard the K_i located on the upper levels as an approximation, compressed version, model or even filter of the D_i. Other viewpoints and interpretations are certainly possible and sensible in different contexts. Furthermore, the sample can not only be used for data analysis but also be visualised straight away. Scatter plots and approximate density plots [30] are natural candidates for interactive data exploration and have proved useful as starting points for identification of regions of interest and of clusters. However, the contribution of this paper is the interpretation of the sample as the result of a uniform random process and the subsequent use as a equi-depth histogram.

Scientific data very often come as multi-dimensional data. However, B-Trees in their basic form index only one-dimensional data, *i.e.*, they only index a single attribute. Yet by using surrogate keys $K_i = f(D_i)$ researcher have extended the B-Trees to work on multidimensional data as well [26, 29]; the resulting data structures are called z-kd-B-Trees or UB-Trees. [15] present an overview and classification of many multi-dimensional data structures.

The structure of this paper is as follows: After reviewing some more related work we sketch the preliminaries used in this paper. We then present the basic tree-traversal algorithm for extracting a sample. The remaining sections of the paper outline the use of the data structure in our visualisation system 3DVDM (3-Dimensional Visual Data Mining) [7] and analyse the algorithm performance, both in terms on resource consumption and accuracy. The final section summarises the contributions of the paper and outlines directions of future research.

2 Related Work

The classic reference for the B-Tree is [5]; [29] extend this work for multi-dimensional data by linearising the data space using a space-filling curve, the z-transform. For a discussion of implementation issues related to B-Trees, see [17]. Related Work can be divided in to several categories: Maintaining (approximate) statistics of data is a recurring problem in data warehousing. For example, [27] present a method of maintaining statistics extending the R-Tree with special, predefined aggregation information. [25] investigate algorithms for random sampling from B+ Trees.

In the areas of parameter estimation, histograms, and statistics in databases there are many works avail-

able. [33] provides a very readable introduction especially to density estimation and histograms. In [1], the authors present a proposal for data-dependent and space-efficient histograms. [24] and [23] are examples of approaches to gather statistics mainly for query optimisation in relational databases. These latter approaches are static, *i.e.*, the data set has to be re-scanned after updates. In contrast, since our method re-uses the B-Tree, which changes shape as the data are updated, there is no need to ever scan the complete data set. Consequently, it does not require additional memory resources. Furthermore, the quality guarantees are data-independent. While [24] can also be used to sample data, [1] only consider histograms. These approaches are also different in spirit since they are general mechanisms that require additional resources. Our approach mostly re-uses existing data structures. An additional feature is that it does not introduce aggregate values reflecting the data distribution in the interior nodes of the tree as introduced by [3] where sampling from arbitrary pseudo-ranked B+-Trees is investigated.

There is also a plethora of work in visualising and browsing data sets that is relevant in our context. We refer to [14] and [32] for overviews and also detailed coverage of a number of topics. We refer to more related work when appropriate throughout the paper.

3 Preliminaries

This section recalls some of the basic definitions we need throughout this paper. We write X and Y for arbitrary random variables, and K for a random variable containing key values and D for a random variable containing data values, describing key/data pairs in dictionary data structures.

Definition 1 *Let K be a discrete random variable and $f(K_i)$ the corresponding probability function. Then $K : F(x) = \sum_{K_i \leq x} f(K_i)$ is called the density function.*

The K_i that appear in the above definition are actually the same K_i that are inserted as keys into the B-Tree. The following definition is adapted from [29]:

Definition 2 *Let $a_1, \ldots, a_n$ be a tuple of bitstrings (i.e., strings consisting of the symbols* 0 *and* 1*) of length m. Then the bitstring*

$$zt(a_1, \ldots, a_n) = a_{1,1}, a_{2,1}, \ldots, a_{n,1}, \ldots, a_{1,m}, a_{2,m}, \ldots, a_{n,m}$$

is called the z-transform *of $a_1, \ldots, a_n$, where $a_{i,j}$ indicates the j-th symbol of the bitstring a_i. We call $a_{1,i}, a_{2,i}, \ldots, a_{n,i}$ a* run *of the bitstring.*

Informally, the z-transform of a sequence of equi-length integers is computed by interleaving their bit representations. Note that we restrict the tuples to be unsigned integers; see [29] for a discussion of why this is sensible and how to handle other datatypes. Figure 1 displays an example of a two-dimensional UB-Tree

For this paper, we assume that the reader is familiar the B-Trees and their most important variants [9, 10]. However, for our purposes we slightly alter the basic definition of B-Trees [22] to capture more implementation features.

Definition 3 *A* B-Tree of order *m is a tree that satisfies the following properties: (1) Every node has at most m children. (2) Every node, except for the root and the leaves, has at least $m/2$ children. (3) The root has at least 2 children (unless it is a leaf). (4) All leaves appear on the same level, and carry no information. (5) A non-leaf node with $i+1$ children contains i keys. Furthermore, we assume that all keys and associated data are of the same byte length.*

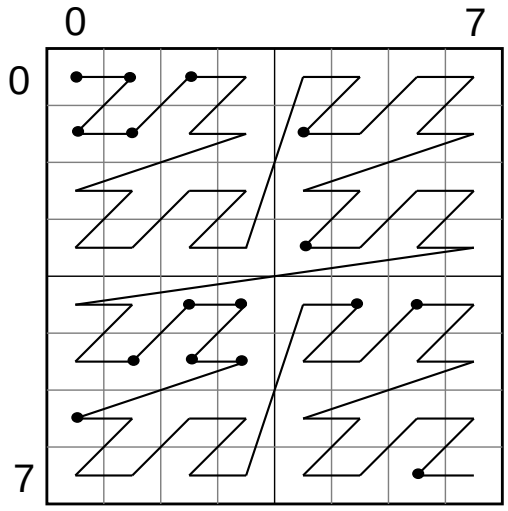


Figure 1: Example two-dimensional UB-Tree

For our purposes a UB-Tree is a B-Tree into which tuples $\langle f(zt(K_i)), D_i \rangle$ are inserted. Furthermore, we assume that the keys in a B-Tree index a file containing records $\langle f(K_i), D_i \rangle$. Following [33] we furthermore define the following:

Definition 4 *Let x_0 be an origin, b a real number, the* bin width*, and n an integer, the* number of bins*. Then the intervals $[x_0 + mb, x_0 + (m+1)b)_{0 \leq m \leq n-1}$ are called the* bins *of an* equi-width histogram*. Furthermore,*

$$h(x) = \frac{1}{nb}(|X_i \text{ in the same bin as } x|)$$

is called an (equi-width) histogram*. If we allow b to be a function of m, then we call the intervals $[x_0 + mb(m), x_0 + (m+1)b(m+1))_{0 \leq m \leq n-1}$ such that each interval contains (approximately) the same number of X_i then we call*

$$h(x) = \frac{1}{n} \cdot \frac{|X_i \text{ in same bin as } x|}{|\text{width of bin containing } x|}$$

an equi-depth histogram*.*

Equi-depth histograms are used for selectivity estimation in query optimisation in [28]. When presenting density information graphically, one often uses kernel-estimated density functions:

Definition 5 *The* kernel estimator *with kernel K is defined by*

$$f(x) = \frac{1}{nh} \sum_{i=1}^{n} K(\frac{x - X_i}{h})$$

where h is the window width (alias smoothing parameter or bandwidth) and K a kernel function*, i.e., a function such that*

$$\int_{-\infty}^{+\infty} K(x)\, dx = 1.$$

Most of the plots in this paper use kernel estimation techniques for smoothing and removing bias from example distributions. Furthermore, kernel estimation is useful for visualising and post-processing of samples obtained by the methods we describe in this paper.

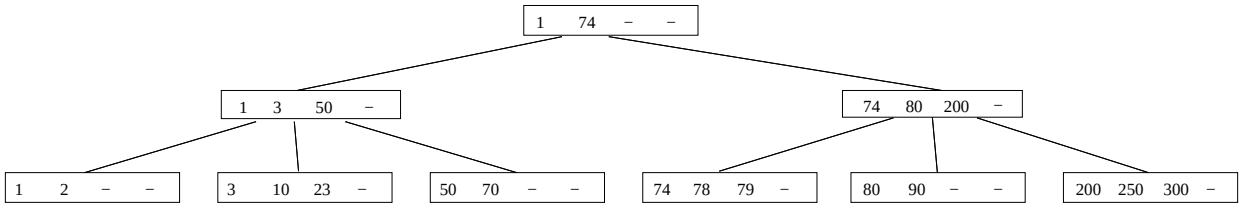


Figure 2: Example B-Tree

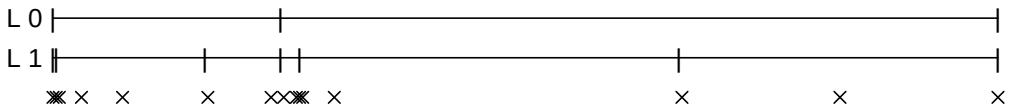


Figure 3: Level-wise space partitioning implied by example B-Tree in Figure 2

4 Algorithms (U)B-Tree

This section presents algorithms to extract a sample from a B-Tree and describes the bulkloading of one- and high-dimensional data in our framework. We also look at the accuracy of the sample for bulk loading pre-sorted data. Note that we do not consider issues like data cleansing and other pre-processing which in general are necessary for scientific data.

4.1 The One-Dimensional Case

In the case of one-dimensional data, *i.e.*, data where a single key attribute is used as a primary key, the basic data flow looks as follows:

$$\textit{source data} \rightarrow \textit{key extraction} \rightarrow \textit{insertion} \rightarrow \textit{traversal} \rightarrow \textit{interpretation}$$

We first introduce the core ideas presented in this paper informally by considering the following example and Figures 2 and 3.

Example 1 *Let K be a random variable and*

$$K_i : \{1,2,3,10,23,50,70,74,78,79,80,90,200,250,300\}$$

be observations. Inserted as $\langle K_i, \emptyset \rangle$ *into a B-Tree, they result in a tree of a shape similar to the one depicted in Figure 2; the height of the tree is three. Thus, there are two interior levels: level 0 is the root page, level 1 the values stored just beneath the root. Figure 3 illustrates the shape of the tree by projecting the space partitioning implied by the keys found on levels 0 and 1 on the domain axis. L0 denotes level 0, the root, L1 the first level, level 1, of the tree. The crosses in the lowest line are a scatter plot (see Section 5.2) of the source data.*

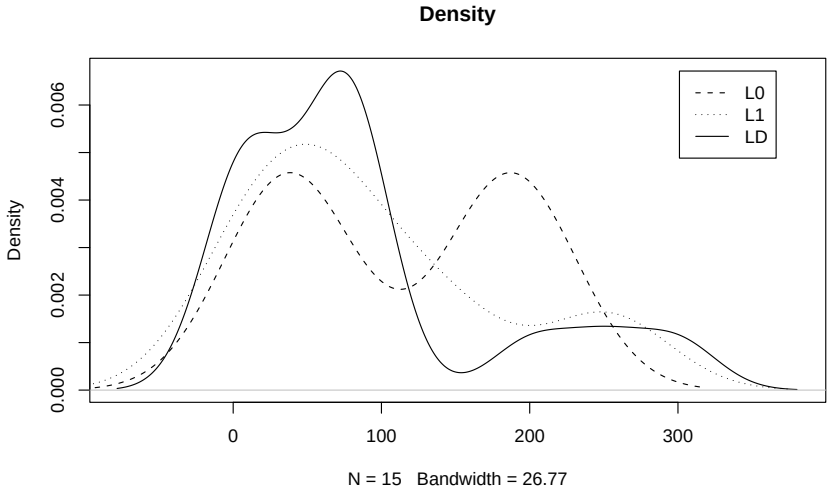


Figure 4: Kernel-estimated Probability Density Function

traverse $p\ l\ c : page \to int \to int \to \{int\}$
  p : *current page*
  l : *level in B-Tree to be read*
  c : *current level in B-Tree* $\{$
  case
    $c = l$: **return** $\{$*keys located on* $p\}$
    $c > l$: **return** $\emptyset$
    $c < l$:
      let $m = number\_of\_keys(p)$ **in**
      return $\bigcup_{i=0}^{m}(\textbf{traverse}(key[i] \to page, l, c+1)$
$\}$

Figure 5: B-Tree traversal algorithm

We now can compute the Probability Density Function (PDF) implied by each level the tree (LD denotes the data level). They are show in Figure 4. In this figure, the motivation for our work becomes clear: the first level already exhibits parameters similar to those of the source data – even for a toy example with far too few data points like this one.

In the following subsection we observe that for a suitably bulkloaded data set there exists an integer k such that every k-th key extracted from the original data set D is present on a certain level of the tree. Later in Section 6 we generalise this statement to trees obtained from random insertions. The key observation is that the sample obtained by the traversal algorithm can be considered the result of unbiased uniform sampling. It is thus suitable for a broad range of practical applications.

Pseudo-code for the traversal algorithm is depicted in Figure 5. The notation $key[i] \to page$ denotes the disk page containing the successors of $key[i]$ in the appropriate interval. Note that the output of the traversal is sorted and that the algorithm is free of side-effects. Also, many implementation details like locking are not included in the algorithm for the sake of presentation clarity.

4.2 Optimality for sequential bulk loads

The following corollary shows that the algorithm depicted in Figure 5 is optimal for certain important special cases. In this subsection we assume that the behaviour of the B-Tree strictly adheres to all points of Definition 3. We observe that when we bottom-up bulkload a B-Tree according to [20, 38] with data sorted according to the keys, the sample of K as returned by the algorithm displayed in Figure 5 exhibits the following property:

Corollary 1 *For a bottom-up constructed B-Tree of height greater than one there exists an integer k_i such that every k_i-th element of a sorted data set D is found on level i where i is a B-Tree level.*

When we do not refer to a particular level, we also write k instead of k_i. Intuitively, the corollary is true because we can predict exactly which keys will be propagated up when a page overflows. Recall that we assume that all keys are of the same size. This even holds for neighbour merging. When bulkloading sorted data the data stream is effectively divided up into bins, each containing the same number of data. Thus, when a leaf node overflows, exactly one node per bin is moved a level up. Assuming that the split algorithm chooses the key to the propagated up by its index number, the process is data-independent and deterministic.

We now turn our attention to k of the above corollary. The minimum number of keys on level i is $2 \cdot t^{i-1}$ if $i > 1$ and 1 if $i = 1$, the maximum number t^i [10], $t = m/2$ being the minimum fan-out (which exists since we assume constant key length and no compression). Using these bounds we can estimate the value of k on level i:

$$\left\lfloor \frac{|D|}{t^i} \right\rfloor \leq k \leq \left\lceil \frac{|D|}{2t^{i-1}} \right\rceil$$

Furthermore, in Section 6 we present an implementation-dependent method for calculating k for sorted bulkloads. We are not certain if this property still holds if advanced bulkloading algorithms with delayed insertions like buffer trees are used [4]. The other extreme, when data are inserted and deleted randomly, can be considered the worst case. In this case, we have to take into account a certain degree of fuzziness. Later, in Section 6 we will try to illustrate this and present an experimental quantification which shows that even then the sample can still be considered uniform. Note that we are solely reasoning about the positions of the data, *i.e.*, the fact that a K_i is a k-th element in the data set, independent of the data themselves. This implies that our estimates hold for any data distribution.

5 Applications in Visualisation

In the previous section, we have seen how to draw samples with the help of B-Trees. In this section, we focus our attention on our primary application context, the visualisation of large data sets such as those typically occurring in scientific computing. For this section, we also assume that all data are normalised to the unit interval.

5.1 The Multi-Dimensional Case

Data in visualisation applications are usually multi-dimensional rather than one-dimensional. Therefore it is necessary to generalise the techniques discussed so far to the multi-dimensional case. Fortunately, it does not differ greatly from the one-dimensional case with respect to bulkloading and the traversal algorithm and requires only small extensions. However, the algorithms and the program flow have to be generalised when data are to be bulkloaded and the result of a traversal is to be interpreted.

```
volume b n c : bitstring → int → int → float
  b : bitstring encoding a z-value
  n : number of dimensions of b
  c : current step {
    if length(b) = 0
    then return 0
    else {
      let v = first run of b in /* see Definition 2*/
      let v' = rest of b in /* all of b but the first run */
      let s = 2^(-c·n) in
      return vol(v) · s + (volume v' n (c+1))
    }
  }
```

Figure 6: Computing the volume of a z-interval

Technically, the program flow in the case of high-dimensional data features two extra steps for encoding and decoding that data:

$$\textit{source data} \rightarrow \textit{key extraction} \rightarrow \textit{z-encoding} \rightarrow \textit{traversal} \rightarrow \textit{z-decoding} \rightarrow \textit{interpretation}$$

The encoding of the data is straight-forward by implementing the formula given in Section 3. Its time complexity is merely linear in the size of the input. Decoding works in a similar manner. Since the formula given in Definition 2 is lossless, it is possible to decode the data uniquely. The implementation is straightforward as well [29].

Construction of an equi-width histogram. It is interesting to see how an equi-width histogram can be constructed from a set of z-encoded data. Since most statistical packages operate on Euclidean data rather than z-transformed data, it is frequently necessary in daily work to switch from one space to the other. To construct the histogram, we interpret the sample as a set of intervals; since it is already sorted, this requires no additional effort and is easily done by interpreting neighbouring pairs of points as start and end points of intervals. The basic idea is now to compute the volume of the individual intervals and map them to densities in the Euclidean target space. Suppose we are given a z-encoded point in n-dimensional space $a = a_{1,1}, a_{2,1}, \ldots, a_{n,1}, \ldots, a_{1,m}, a_{2,m}, \ldots, a_{n,m}$. To calculate the histogram, we use the density information encoded in the intervals as a starting point. To do this we need to know the volume of the interval $[\overline{0}, a)$, *i.e.*, the space described by the z-curve starting at zero and leading to a. Then we can identify a set of hypercubes each of volume 2^{-kn} with every run $a_{1,k}, a_{2,k}, \ldots, a_{n,k}$.

The algorithm for calculating the histogram shares some core ideas with the range query algorithm described in [29]. In the following, let $vol(b)$ denote the number of hypercubes covered by b. For two dimensions, a lookup table could look as follows: $\{00 \rightarrow 0, 10 \rightarrow 1, 01 \rightarrow 2, 11 \rightarrow 3\}$. Then the algorithm in Figure 6 calculates the space occupied by the interval $[0, b)$ in the z-space.

To calculate the space occupied by an interval $[z_1, z_2)$ one applies the algorithm in Figure 6 twice and computes $\textbf{volume}(z_2) - \textbf{volume}(z_1)$. Now, being able to calculate the volume of a z-region, *i.e.*, the space bounded by two z-encoded points, one can calculate the densities implied by the keys in the B-Tree. In general, it is impossible to know the exact number of keys in this region. However, we can assume uniformity and thus arrive at reasonably accurate estimates. Note that if the data were inserted in sort order, we even can compute the exact number of elements in the interval. In practise, we can

resort to the solution that we regard the keys we find on a level of the tree as a sample of the complete data space and thus do all our calculations with respect to the sample space. To derive an n-dimensional equi-width histogram, we first note that the z-encoding divides the n-dimensional space into disjunct sets of hypercubes. Suppose now we are given an n-dimensional hypercube at the same resolution as the z-transform, namely $2^{number\ of\ steps}$ intervals along each axis. We then sort the sequence $z_1, \ldots, z_m$ as if it was all integers and traverse the resulting sequence interpreting adjacent z_i, z_{i+1} as intervals $[z_i, z_{i+1})$. Then, we calculate $\textbf{volume}(z_i) - \textbf{volume}(z_{i+1})$ and assign this fraction to every bin. From these intensities we can then derive densities and histogram counts, whichever is desirable.

If, as in our case, the resolution of the histogram matches the resolution of the z-transform, the transformation is lossless. However, if the resolutions don't match, it is necessary to make compromises to alleviate the effects of discretisation. Since this topic has been investigated to great detail by the statistics community we refer to the literature for a number of solutions like kernel estimation [33].

Note that the parameter k in Corollary 1 for high dimensional data may become rather large on levels close to the root since the key length is usually linear in the number of dimensions and only few keys may fit on a page. Unfortunately, for performance reasons one would prefer traversing as few levels as possible since descending from one level to another (*cf.* the recursive call to **traverse** in Figure 5) may involve random I/O and thus incur high costs. This implies that approximations and samples may not be representative enough to enable informed reasoning and, at the same time, drawing more samples may impede on interactivity. However, we remark that this a problematic area in general and that the problems caused by large data volumes with little information in high-dimensional data management are not related to our method in particular. We refer to [6] for a quantitative and qualitative analysis of the problem.

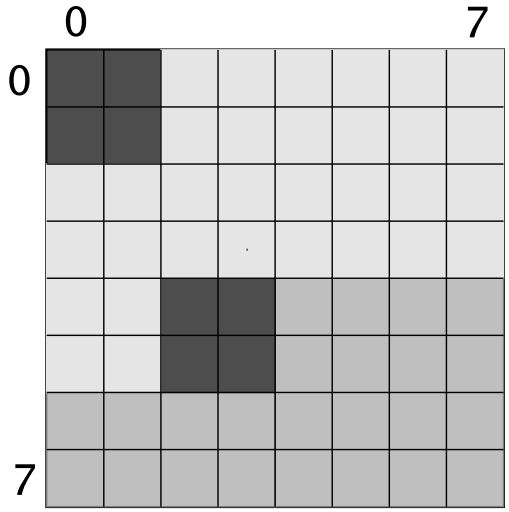
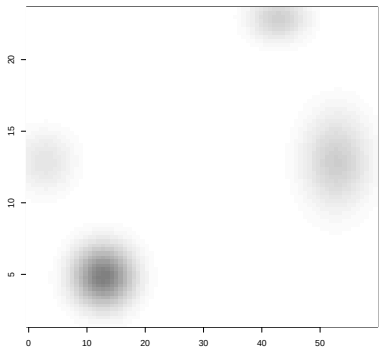
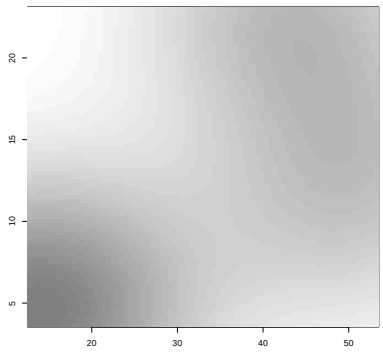


Figure 7: Approximate densities of Figure 1

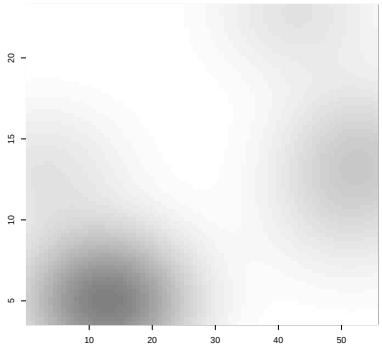
In visualisation of scientific data sets, density estimation is an important but by no means the only interesting topic. Frequently, estimation of further parameters like mean, standard deviation is required either on a global or, more realistically, on a local basis. These results may then be used to remove outliers, build synopses or estimate kernels. See [30] for a well-written textbook on density estimation and advanced visualisation of scientific data sets. We should also mention that many of the problems we address in this paper also play a role in Decision Support Systems (DSS) [18] and On-Line Analytical Processing (OLAP) [19]. There are also database management systems that were designed with efficient data mining support in mind, *e.g.*, [8], which try to tackle data processing in scientific applications from novel angles.



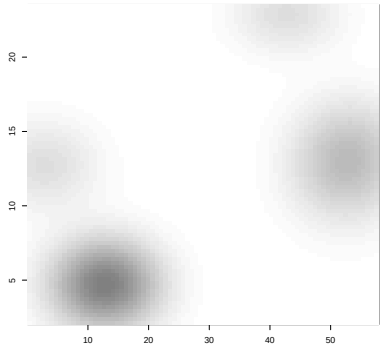
(a) PDF of source data



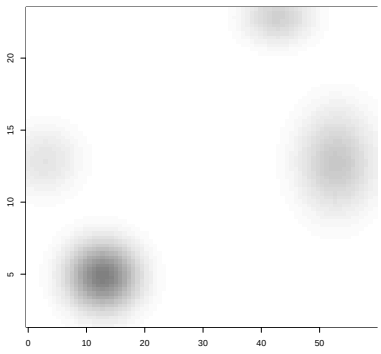
(b) PDF in B-Tree root



(c) PDF in B-Tree level 1

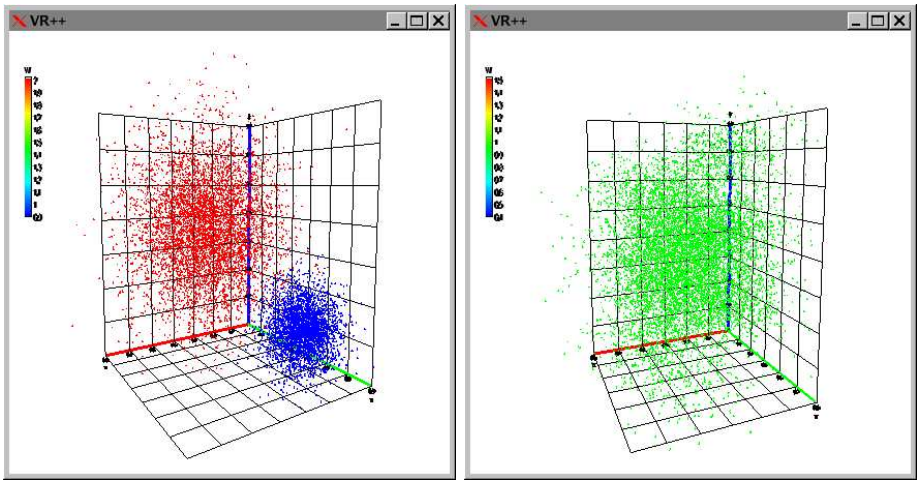


(d) PDF in B-Tree level 2



(e) PDF in B-Tree level 3

Figure 8: Two-dimensional computed PDFs



(a) Zooming to two clusters (b) Zooming to a large cluster

Figure 9: Screenshots of the 3DVDM System

5.2 Computation of Density and Scatter Plots

In this subsection, we examine how to use the samples to obtain two basic types of visual representation: density plots and scatter plots.

5.2.1 Density Plots

We use density plots which approximate the number of points in a histogram bin with colours. For example, grey-scale density plots might encode the highest densities with the colour black and low-density areas with white. Figure 8(a) displays an error-free density plot computed from a large data set without using B-Tree sampling. In contrast, Figures 8(b)–(e) make use of our method. They were computed using samples drawn from the root level and levels 1 to 3. Note how the plots converge to the above error-free plot indicating the different approximations and precisions.

In our setting, there two basic ways to obtaining the input data to density plots. On the one hand, one could take the sample obtained by our method and use it directly as input to a density approximation algorithm. On the other hand, it is also possible to view the sample as a set of intervals and, based on this knowledge, calculate densities directly. Which of these approaches is preferable, probably depends on the specific application. In term of software complexity, especially for multi-dimensional data the interval approach requires some careful engineering but also adds some possibly useful knobs for tuning applications. Figure 7 shows how the first method can be applied to the example in Figure 1. We assume that four points fit on one B-Tree page.

5.2.2 Scatter Plots

In the case of scatter plots, we do not have so many choices since we are not interested in regions but in points only. The only decision to take is whether to use the sample directly or use interpolation. The latter approach has the disadvantage that we may actually plot points that are not part of the original data set. Thus, for our applications we use the first interpretation.

Figure 9 shows two examples of scatter plots in 3DVDM [7]: Scatter plots are a very common tool for visualising point data. Technically, in their basic form they use one point in the diagram for one data point. It is thus natural for us to use the sample interpretation and not the interval interpretation. There are a number of challenges to be met when one wants to produce good scatter plots, the most prominent being overplotting, *i.e.*, using so many data points that the diagram exhibits jittering and does not reflect the real number of points. For an in-depth discussion of these and related issues see [30].

5.3 Parameter Estimation

Relational Database Management Systems offer a number of standard aggregate functions like **min**, **max**, **sum**, **count**, and **avg**. It seems that **avg** is the most interesting one in our context since it subsumes **sum** and **count** while **min** and **max** are not specific to our method.

Computing the Mean. In the following, we thus turn our attention to the computation of the arithmetic mean. We mainly try to answer the questions: How fast can we expect the computation to converge? What are the error bounds? Note that in the following, we again assume that values are normalised in the unit interval. The common definition of the arithmetic mean is (n denotes the cardinality of the data set D):

$$\mu(D) = \frac{1}{n}\sum_{i=1}^{n} D_i$$

We now view a level i in a B-tree as a set of interval ranges

$$[K_1,K_2),\ldots,[K_i,K_{i+1}),\ldots,[K_{m_i-1},K_{m_i}).$$

We write R_j to denote the middle value of the interval range $[K_j,K_{j+1})$; and $R_j^+ = K_j$ denotes the start point, whereas R_j^- denotes the end point K_{j+1} of the interval R_j. Thus, if $R_j = [R_j^-,R_j^+)$ then $\bar{R_j} = (R_j^+ - R_j^-)/2$. Recursive application of this rule can be used to compute the mean on a B-Tree level i denoted by L_i. Thus, we have

$$\mu_{est}(D,L_i) = \frac{1}{m_i}\sum_{j=i_0}^{m_i} R_j.$$

The error of the estimated mean now depends on the level i:

$$\begin{aligned}
\varepsilon(\mu,D,L_i) &= |\mu(D) - \mu_{est}(D,L_i)| \\
&= |(\frac{1}{m_i}\sum_{j=1}^{m_i}\frac{m_i}{n}\sum_{D_i\in R_j} D_i) - (\frac{1}{m_i}\sum_{j=1}^{m_i}\bar{R_j})| \\
&= |\frac{1}{m_i}\sum_{j=1}^{m_i}(\frac{m_i}{n}\sum_{D_i\in R_j} D_i - \bar{R_j})|
\end{aligned}$$

Let $|R_j|$ denote the length of the interval R_j, *i.e.*, $|R_j| = R_j^+ - R_j^-$. Then

$$\begin{aligned}
&\leq \frac{1}{m_i}\sum_{j=1}^{m_i}\frac{1}{2}|R_j| \\
&\leq \frac{1}{2m_i} \leq 1/m^i \qquad (m \text{ as in Definition 3})
\end{aligned}$$

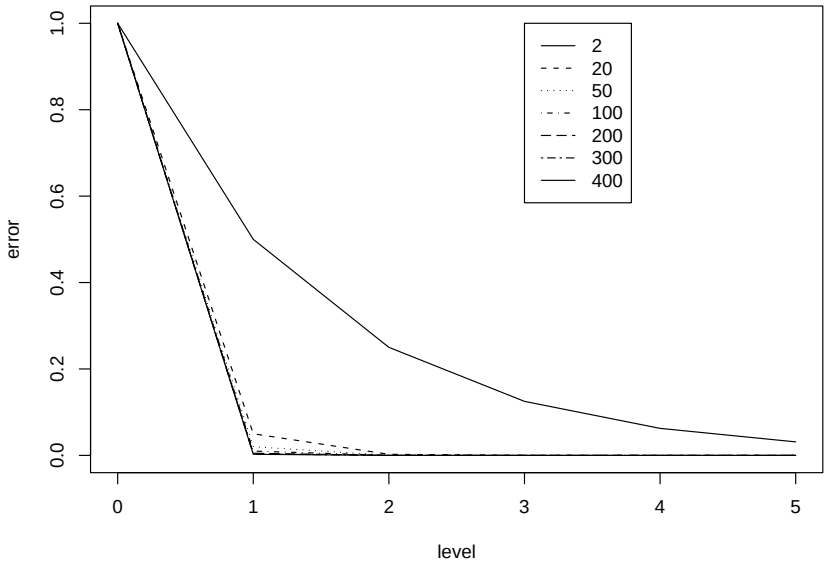


Figure 10: Error for Different Fanouts

Figure 10 visualises the error estimate depending on the maximum number of children m of the tree. As expected, it converges quickly thus demonstrating the potential of the method. We illustrate the convergence with examples in Section 6.

5.4 Other Applications

According to [1], 'a major disadvantage of histograms is the cost of building and maintaining them.' The samples extracted by our method can be viewed as histograms that are automatically updated. However, note that in general settings some of our underlying assumption don't hold anymore. Users may insist on having prefix compression, variable-length keys, *etc*. – features that possibly impact the quality of the histogram or sample. Without making basic assumptions about not only data distribution but also other data characteristics like prefix-compressibility, it is hard to estimate the effects. Additionally, delayed deletions and other optimisations may add further uncertainty. However, we consider a detailed analysis a topic for future work.

A straight-forward application is removal of outliers. When an interval has a large volume compared to the other intervals, it does not contain many points. In applications that are mainly interested in clusters, these regions can be dropped immediately and need not be analysed.

6 Implementation and Experiments

This section describes some of the problems and opportunities we encountered while implementing the method presented in this paper and while applying it. We also discuss some performance figures that illustrate the behaviour in the system 3DVDM [7].

6.1 Implementation Issues and Techniques

We used Sleepycat's Berkeley DB [34] to implement the level-wise traversal on an industrial strength B-Tree. Since Berkeley DB does not provide this functionality natively, we had to add new routines to the database kernel. However, we were able to design the new routines along the lines of already existing functionality. In particular, it was not necessary to add any new data structures to Berkeley DB; the inner workings remained unmodified. Since Berkeley DB is written in C, this language was also our implementation language.

We turned off prefix compression for two reasons: (1) Since prefix compressed relations only store the part of a key that is needed to distinguish it from its neighbours in the tree (according to the comparison function on the appropriate datatype), we would lose information that is absolutely necessary for our estimation. (2) In extreme cases prefix compression can significantly increase the fan-out of the B-Tree so that our algorithm would produce biased samples. While, in our experience, the effect of (2) is more or less negligible on numerical data, it might significantly alter the well-behavedness of the estimation algorithm if different physical encodings or other datatypes are used.

We went about extending Berkeley DB with the appropriate algorithms by rewriting the built-in lookup function for B-Trees. It was also necessary to implement custom cache management to ensure that the traversal consumes only minimal resources and does not cause unnecessary stalls for other processes. Depending on how many levels of the B-Tree are pinned in the disk-page cache, it might be advantageous to apply techniques similar to those investigated in [31] to improve performance.

So far, we have assumed that the B-Tree is used as a primary index. This implies that the index is sparse and cannot contain duplicate keys. Should we want to accommodate duplicates we would have to switch to a dense index – a technique usually employed to implement secondary indexes. The interior nodes of a sparse primary index would then not be suitable for our purposes since the sortedness of the underlying file is usually used for implementing sparsity. So even if a there are a huge number of copies of a key K_i in a sparse index, there would only be one instance of K_i in the interior nodes of the B-Tree. However, our techniques also work with dense or secondary indexes without modification. Since both kinds of indexes are usually implemented in an RDBMS (as well as in Berkeley DB) we do not discuss the case of duplicate keys. However note that the building of a secondary index especially on non-sorted data may incur significant storage overhead and performance penalties.

For bulkloading the index [38] we recommend using sorted data or nearly sorted data to avoid random I/O. In the case of large amounts of data that exceed the size of the main memory it proved useful to pre-sort the data; in the case of multi-dimensional z-transformed tuples the conversion can be done in chunks that fit in main memory and, once residing in main memory, the tuples can be sorted according to the key values and the comparison function.

Note that in our scenario we computed the keys K_i from the data D_i rather than declaring a subset of the D_i as keys to make the process more flexible and allow for an easy integration of z-transformed keys. Most modern database management systems such as DB2 [11], Oracle [13] and MS SQL Server [12] provide support for *function-based* B-Trees. Since scientific data usually undergo extensive data cleansing and preprocessing, the function computation in this step can be naturally included into the pre-processing phase of the raw sensor data. So, even if a DBMS does not provide support for function-indexes an implementation is still possible – at the cost that a tuple cannot be updated anymore using plain SQL [2]. However, since scientific database are often read-only anyway this does not appear too severe a restriction. Nevertheless, the tree-traversal algorithm itself is still a separate implementation issue.

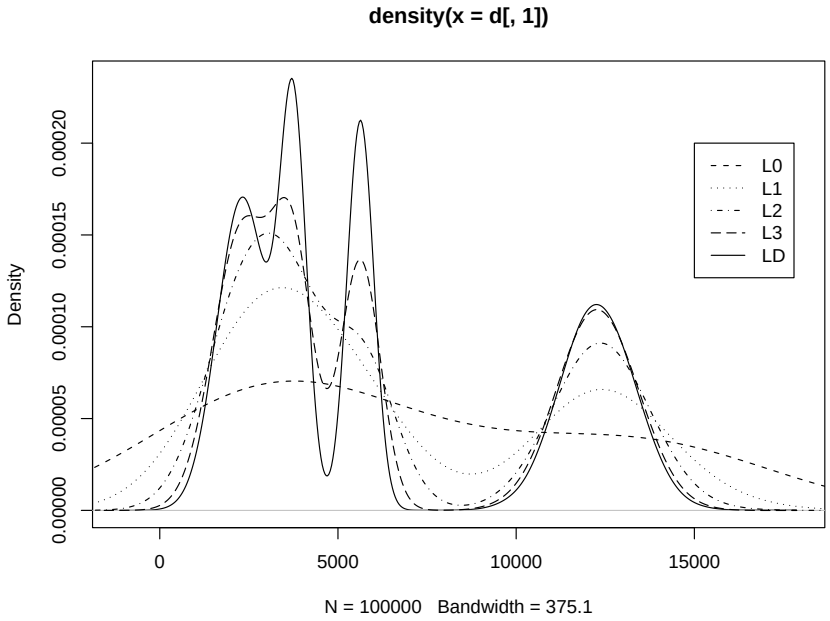


Figure 11: Kernel-estimated PDF

6.2 Experiments

All of the experiments were carried out on a Linux PC featuring Redhat 7.2, a Pentium III (Coppermine) CPU running at 1 GHz, 512 MB of main memory, and an IDE Western Digital WD400BB hard drive.

The following table illustrates the cost of our method compared to sequential scan on a table comprising more that one billion tuples (page size is 8KB, key size 10 bytes):

level	B-Tree	sample size	sequential scan
0	$<0.001s$	32	
1	$0.01s$	9424	$4750s$
2	$120s$	2949852	

Because of the small sample size, level 0 is not really useful. However since the first level of the tree is kept in cache, it is very cheap to obtain a sample of already nearly 9500 points. Accessing level 2 requires random I/O making the additional step expensive but is still significantly cheaper than a sequential scan that would be required by other sampling methods like [24].

For another experiment, we build on Example 1. Figure 11 shows the probability density functions (PDFs) of the samples drawn from the different levels. The distribution is synthetically generated and features extreme densities. For example, the slim middle peak on the data level (LD) has a standard deviation of just 0.0001. The figure demonstrates both the limitations of sampling in general as well as the convergence of our method. Note that the PDF has been smoothed by the kernel estimator displayed in the figure.

Figure 12 refers back to the density plots of Subsection 5.2.1. It exemplifies convergence in a two-dimensional setting. Again, estimates are rough in the beginning but converge quickly.

level	$\mu(X)$	$\sigma^2(X)$	$\mu(Y)$	$\sigma^2(Y)$
L0	32.58	318.32	11.91	68.68
L1	26.53	374.42	10.03	36.82
L2	27.95	405.03	10.26	35.45
L3	28.16	396.95	10.54	38.04
data	28.00	392.99	10.45	37.83

Figure 12: Convergence for plots in Section 5.2.1

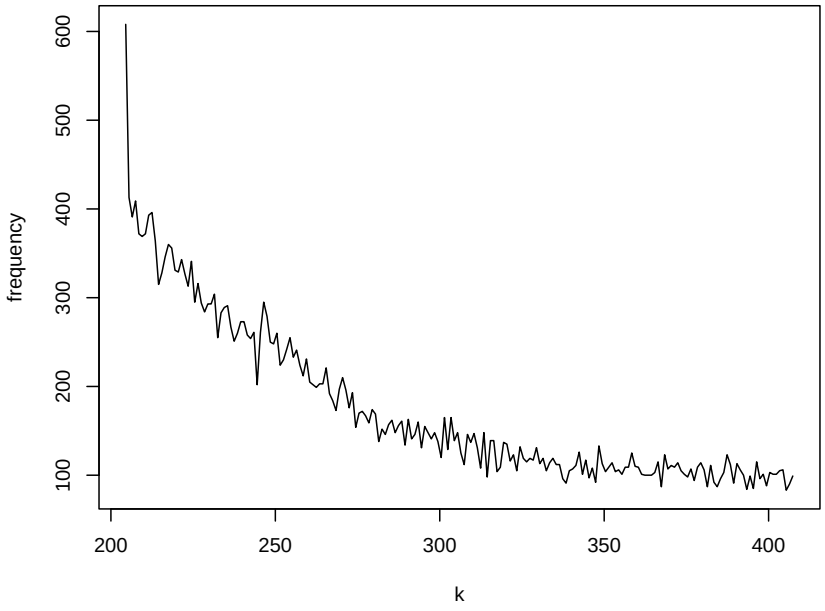


Figure 13: Histogram of *k* for random insertion and page size 8192

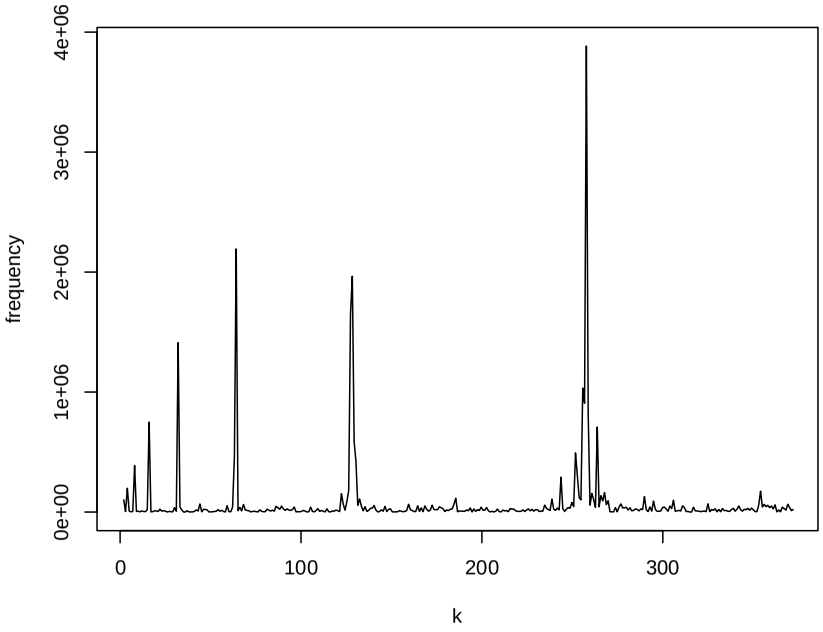


Figure 14: Page size 8192 unsorted data

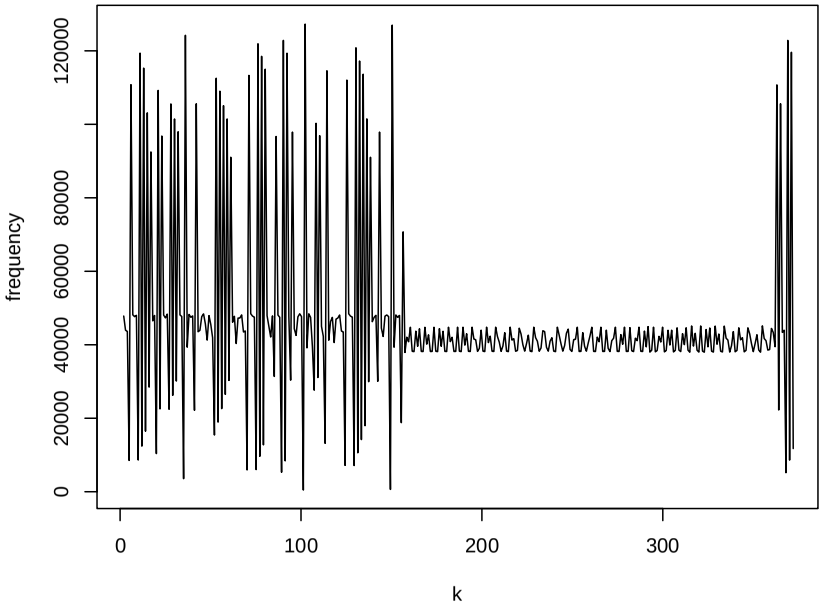


Figure 15: Page size 8192 sorted data

6.3 Sampling Quality

In Section 4.2, we observed that if a B-Tree of equal length keys is bulkloaded with sorted data then there exists an k such that every k-th key in the source data is included in the sample computed by our method. We now investigate why this is so by studying our implementation; thereafter, we also try to see what effect random insertions instead of sequential bulkloading have. This investigation is important since it underlines the robustness of our method.

Bulkloading B-Trees with pre-sorted data [38] is done in a bottom-up fashion. From the sequence of D_i ordered subsequences of K_i are extracted and packed into disk pages. This process repeats recursively. In our context, we identify disk pages with histogram bins. The bottom-up methods chooses *one* element from every bin, usually the first one. Since, for every individual level of the B-Tree, the same number of elements fit on one page, there exists a K_i for every level i such that each level contains every k_i-th element from the sorted input. Thus, it is even possible to come up with an analytical way of calculating the k_i: let n_i be the number of keys that fit on a page on level i and let l be the number of levels in the tree. Then k_l on the lowest level equals n_l, and in general

$$k_i = \prod_{j=i}^{l} n_j.$$

Note that the n_i may indeed be different on the different levels due to implementation strategies or because it is desirable to adjust the page fill factor to the depth of a node. For a page size of 8 KB and a key size of 10 bytes, we found n_l, the leaf level, typically to be 339 and $(n_i)_{i<l}$ to be 313 in the interior levels. However, these numbers are implementation dependent.

We now look at the sampling quality for random insertions. Another perspective of looking at the n_i is to regard the choice of elements that go into the leaf level of the tree as a uniform sample. The interior level now represent subsamples of uniform samples, and, thus, are in return uniform samples themselves. Exponential distributions are used to model the distance between events with uniform distribution in time [35]; they are often defined by their probability density function

$$F(x) = a \cdot e^{-a \cdot x}.$$

Figure 13 displays a typical distribution of values of k on a page size of 8 KB and key size 10 bytes.

From the shape of the curve, we conjecture that we can model the indexes of the keys on a level as a uniformly distributed random variable. Thus we can interpret the exponential distribution of k shown in the above figure in the following way: For uniformly distributed random insertions of D, the distances between the indexes of the D_i feature an exponential distribution. Thus, the randomness of the insertions is propagated up to the keys on interior nodes and levels of the tree. In this sense, we conclude that our method again delivers good, unbiased uniform samples independent of the actual data distribution or insertion order. A final remark on why we chose uniformly distributed random insertions: In terms of entropy, they are the worst-case for B-Trees that are optimised for sequential bulkloads and thus present the greatest challenge.

The two figures that conclude this section are based on the following consideration: the number of keys in the root page on the tree, greatly impacts how many levels we have to traverse to obtain a sample of a certain size. If the root is likely to contain only few keys, we probably have to descend one level deeper than we might expect. Note that the balance criterion of Definition 3 excludes the tree root.

To underline the viability of this approach, Figure 14 displays a histogram of the number of keys found on the root page during random insertions. It also illustrates the stability of the B-Tree shape and indicates that it, like textbooks tell us, rarely changes. Note that the spikes are logarithmically spaced. In contrast, the effects of sequential bulkloads are displayed in Figure 15. Interestingly, sequential bulkloading makes it more likely to have few keys on the root page. However, in both cases probabilities remain reasonable for our purposes.

7 Conclusion

This paper presented a method that exploits the balance of B-Trees to estimate statistical parameters of a data set by looking at only the upper levels of a B-Tree index. On large data sets the method can outperform linear scan by more than an order of magnitude. We have shown that the sample is always uniform, even in the worst case, and have used the method with great success in visualisation applications.

It is straight-forward to extend relational database systems with the concepts needed to implement our algorithms. In terms of accuracy, we proved that for bulkloads of sorted data, the sample represented by a B-Tree level is very regular in the sense that exists an integer k so that every k-th key is part of the sample. For random insertions, we experimentally demonstrated that the sample is still of predictable and good quality. We applied the method to a number of real-world and synthetic data sets, investigated the features, properties and real-world performance, both in terms of resource consumption and accuracy. We concluded with a review of details and surprises we encountered during the implementation. Additionally, we remarked that our results may prove useful not only in visualising scientific data but also for more standard applications like selectivity estimation, histogram construction.

In terms of future research, we plan to explore the duality of intervals *vs*. samples in more detail and look at the robustness of the sampling *vs*. interpolation approach (*cf*. Section 5.2), especially in the multi-dimensional case. It would be interesting to see whether density estimation algorithms that work on histograms can be used with samples instead of interpolated intervals and what the impact on precision and error estimation is. Furthermore, we plan to investigate the effect of the error bounds we showed in this paper on more algorithms used in the 3DVDM system.

Acknowledgements

The authors would like to express their gratitude towards Gustavo Alonso, Manuel Arrayás Chazeta, Christian S. Jensen, Arturas Mazeika and Zbigniew R. Struzik for comments and discussions.

References

[1] A. Aboulnaga and S. Chaudhuri. Self-tuning Histograms: Building Histograms Without Looking at Data. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 181–192, 1999.

[2] American National Standards Institute. The database language SQL, 1986.

[3] G. Antoshenkov. Random Sampling from Pseudo-Ranked B+ Trees. In *Proceedings of the International Conference on Very Large Data Bases*, pages 375–382, 1992.

[4] L. Arge. *Efficient External-Memory Data Structures and Applications*. PhD thesis, University of Århus, 1996.

[5] R. Bayer and E. McCreight. Organization and Maintenance of Large Ordered Indices. *Acta Informatica*, 1:173–189, 1972.

[6] K. Beyer, J. Goldstein, R. Ramakrishnan, and U. Shaft. When Is "Nearest Neighbor" Meaningful? In *International Conference on Database Theory*, pages 217–235, 1999.

[7] M. Böhlen *et al.* 3D Visual Data Mining. `http://www.cs.auc.dk/3DVDM/`, 2003.

[8] P. Boncz and M. Kersten. MIL Primitives for Querying a Fragmented World. *VLDB Journal*, 8(2):101–119, 1999.

[9] D. Comer. The Ubiquitous B-Tree. *ACM Computing Surveys*, 11(2):121–137, 1979.

[10] T. Corman, C. Leiserson, and R. Rivest. *Introduction to Algorithms*. The MIT Press, 1989.

[11] International Business Machines Corporation. DB2 Product Family. `http://www-3.ibm.com/software/data/db2/`, 2003.

[12] Microsoft Corporation. Microsoft SQL Server. `http://www.microsoft.com/sql/default.asp`, 2003.

[13] Oracle Corporation. Oracle9i Database. `http://www.oracle.com/products/`, 2003.

[14] U. Fayyad, G. Grinstein, and A. Wierse. *Information Visualization in Data Mining and Knowledge Discovery*. Morgan Kaufmann, 2000.

[15] V. Gaede and O. Günther. Multidimensional Access Methods. *ACM Computing Surveys*, 30(2):170–231, 1998.

[16] M. Garofalakis and P. Gibbons. Approximate Query Processing: Taming the TeraBytes. In *Proceedings of the International Conference on Very Large Data Bases*, pages 169–212, 2001. Tutorial Notes.

[17] J. Gray and A. Reuter. *Transaction Processing: Concepts and Techniques*. Morgan Kaufmann, 1993.

[18] J. Hellerstein, R. Avnur, and V. Raman. Informix under CONTROL: Online Query Processing. *Data Mining and Knowledge Discovery*, 4(4):281–314, 2000.

[19] J. Hellerstein, P. Haas, and H. Wang. Online Aggregation. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 171–182, 1997.

[20] T. Johnson and D. Shasha. The Performance of Current B-Tree Algorithms. *ACM Transactions on Database Systems*, 18(1):51–101, 1993.

[21] D. Keim. Databases and Visualization. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, page 543, 1996.

[22] D. Knuth. *The Art of Computer Programming*, volume 3, Sorting and Searching. Addison Wesley, 1998.

[23] G. Manku, S. Rajagopalan, and B. Lindsay. Approximate Medians and other Quantiles in One Pass and with Limited Memory. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 426–435, 1998.

[24] G. Manku, S. Rajagopalan, and B. Lindsay. Random Sampling Techniques for Space Efficient Online Computation of Order Statistics of Large Datasets. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 251–262, 1999.

[25] F. Olken and D. Rotem. Random Sampling from B+ Trees. In *Proceedings of the International Conference on Very Large Data Bases*, pages 269–277, 1989.

[26] J. Orenstein and T. Merrett. A Class of Data Structures for Associative Searching. In *Proceedings of the ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 181–190, 1984.

[27] D. Papadias, P. Kalnis, J. Zhang, and Y. Tao. Efficient OLAP Operations in Spatial Data Warehouses. In *Advances in Spatial and Temporal Databases*, pages 443–459, 2001.

[28] G. Piatetsky-Shapiro and C. Connell. Accurate Estimation of the Number of Tuples Satisfying a Condition. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 256–276, 1984.

[29] F. Ramsak, V. Markl, R. Fenk, M. Zirkel, K. Elhardt, and R. Bayer. Integrating the UB-Tree into a Database System Kernel. In *Proceedings of the International Conference on Very Large Data Bases*, pages 263–272, 2000.

[30] D. Scott. *Multivariate Density Estimation*. John Wiley & Sons, 1992.

[31] B. Seeger, P.-Å. Larson, and R. McFayden. Reading a Set of Disk Pages. In *Proceedings of the International Conference on Very Large Data Bases*, pages 592–603, 1993.

[32] W. Sherman, A. Craig, M. Baker, and C. Bushell. Scientific Visualization. In *The Computer Science and Engineering Handbook*, pages 820–846. CRC Press, 1997.

[33] B. Silverman. *Density Estimation*. Chapman & Hall, 1986.

[34] Sleepycat Software. Berkeley DB Data Store. available at `http://www.sleepycat.com`, 2003.

[35] M. Spiegel. *Theory and Problems of Probability and Statistics*. McGraw-Hill, 1992.

[36] E. Stolte and G. Alonso. Efficient Exploration of Large Scientific Databases. In *Proceedings of the International Conference on Very Large Data Bases*, pages 622–633, 2002.

[37] A. Szalay, P. Kunszt, A. Thakar, J. Gray, and D. Slutz. Designing and Mining Multi-Terabyte Astronomy Archives: The Sloan Digital Sky Survey. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 451–462, 2000.

[38] J. van den Bercken and B. Seeger. An Evaluation of Generic Bulk Loading Techniques. In *Proceedings of the International Conference on Very Large Data Bases*, pages 461–470, 2001.

References including URLs are current as of 12 May 2003.