Managing Statistical Behavior of Large Data Sets in Shared-Nothing Architectures

Isidore Rigoutsos, *Member*, *IEEE*, and Alex Delis, *Member*, *IEEE*

Abstract—Increasingly larger data sets are being stored in networked architectures. Many of the available data structures are not easily amenable to parallel realizations. Hashing schemes show promise in that respect for the simple reason that the underlying data structure can be decomposed and spread among the set of cooperating nodes with minimal communication and maintenance requirements. In all cases, storage utilization and load balancing are issues that need to be addressed. One can identify two basic approaches to tackle the problem. One way is to address it as part of the design of the data structure that is used to store and retrieve the data. The other is to maintain the data structure intact but address the problem separately. The method that we present here falls in the latter category and is applicable whenever a hash table is the preferred data structure. Intrinsically attached to the used hash table is a hashing function that allows one to partition a possibly unbounded set of data items into a finite set of groups; the hashing function provides the partitioning by assigning each data item to one of the groups. In general, the hashing function cannot guarantee that the various groups will have the same cardinality, on average, for all possible data item distributions. In this paper, we propose a two-stage methodology that uses the knowledge of the hashing function to reorganize the group assignments so that the resulting groups have similar expected cardinalities. The method is generally applicable and independent of the used hashing function. We show the power of the methodology using both synthetic and real-world databases. The derived quasi-uniform storage occupancy and associated load-balancing gains are significant.

Index Terms—Hashing, large databases, statistical behavior, uniform occupancy, load balancing, shared-nothing architectures.

1 Introduction

In recent years, there has been a tendency to store and manipulate increasingly larger data sets on networked architectures. We use the term networked architectures to refer to both loosely coupled collections of processors and networks of workstations; the following discussion will assume that we work on shared-nothing architectures.

It has been noted that although the vast majority of the popular data structures are very effective in uniprocessor environments, their decomposition and maintenance in parallel settings is not a straightforward exercise [26], [11], [30], [22]. In this respect, hashing schemes have shown promise owing to the fact that the underlying hash table structure can be shared among several cooperating nodes while at the same time imposing minimal communication and maintenance burden [6]. Whenever a data structure is shared among many nodes and independent of whether they are tightly or loosely coupled, there is always the concern of efficient storage utilization and of course of load balancing during storage and subsequent data manipulation (i.e., insertion, deletion, update) operations.

There are two schools of thought that tackle the problem from different angles, using different premises. In the first approach, the above issues are addressed at the level of the data structure design: The data structure is designed so that the problems relating to efficient data decomposition and storage occur infrequently. The second approach calls for not interfering with the original data structure but rather treating the problem in an orthogonal manner. The methodology that we are proposing here is pertinent to this second category. In particular, we assume that the core data structure is a hash table and is used to access the member items of the database.

Intrinsically attached to each hash table is a hashing function h that in essence allows one to partition a possibly unbounded set $\mathcal{D}$ of data items $\{d_i / i = 1, \ldots, D\}$, into a finite set $\mathcal{G} = \{g_i / i = 1, \ldots, G\}$ of groups: The function h provides the partitioning by assigning each data item to one of the groups. Moreover, both the domain and the range of a hashing function can be either continuous or discrete. Whenever a hashing function h assumes values in a subset of $\mathbb{R}^k$, $k \in N$, then typically a form of quantization is imposed on those values and a subsequent straightforward mapping to an array stride effected, thus allowing for array-based implementations. For the purposes of our discussion, and unless otherwise stated, we will assume that such a quantization is available.

Although conventional hashing techniques perform sufficiently well with small size databases [36], [46], it has become increasingly evident that one must be cautious when using such techniques with very large data sets [31], [39]. High volume data sets make apparent the statistical properties of the employed hashing function. Such properties are manifested by the patterns of utilization demonstrated by the hash table buckets. Nonuniform occupancy of (distributed) buckets leads to nonuniform storage requirements,

- *I. Rigoutsos is with the Bioinformatics and Pattern Discovery Group, IBM T.J. Watson Research Center, Yorktown Heights, NY 10598. E-mail: rigoutso@us.ibm.com.*
- *A. Delis is with the Department of Computer and Information Science, Polytechnic University, Brooklyn, NY 11201. E-mail: delis@poly.edu.*

Manuscript received 29 Aug. 1997; revised 5 June 1998.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number 105566.

ineffective load balancing, and increased query response time. This adversely affects the benefits that one seeks to gain through the use of hashing. The stated nonuniform occupancy of the hash table's buckets has frequently been observed in the past and with databases from different application domains: databases of biological sequences [39] databases of small drug molecules [40], databases of $2-D$ contours [27], and elsewhere [38], [45]. In most of these cases, the employed hash table has been multidimensional. We conjecture that this nonuniformity is not specific to a particular data type, but instead is endemic to all access methods that are built around a hashing scheme.

The nonuniform distribution over the space of invariants results in different lengths for the (distributed) hash table buckets. Since the length of the longest such bucket impacts on the time needed to process the data during a query, nonuniform distribution will adversely affect performance. On the other hand, a uniform distribution not only will reduce execution time but can also result in a much more efficient storage of the hash table structure. Additionally, in a networked implementation of the method, an almost constant occupancy of all the hash table buckets results in an improved load balancing among the processors/workstations [44], [25], [14]. To this point, and to the best of our knowledge, there exists very little published work [9], [37] on determining and exploiting the distributions that one can anticipate.

We can essentially identify three different approaches in the existing bibliography for addressing the general problem of nonuniform utilization of hashing buckets. First, one can increase the dimensionality of the hashing function [5], [31]. Second, use of database-specific heuristics can improve the occupancy pattern of the hash table buckets [24], [8]. Alternatively, Gray-encoding [12], [21] has been used to treat multidimensional arrays as linear spaces that are traversed accordingly; but as such, this approach cannot guarantee optimal performance independently of the access patterns. Finally, a third alternative is the one that is at the center of our work here: We begin with the knowledge of the actual hashing function that the retrieval system uses and derive expressions for the distribution of bucket occupancy over the space of possible indices. We then exploit the derived distribution and show how to effectively improve the performance of the hashing approach, using a two-stage process.

It should be stressed at this point that the method is general and, in principle, is applicable every time that a hashing function is used by a retrieval system *independent* of whether this function is known to the user or not. Indeed, in those cases that the actual hashing function is not known, one can still estimate it using numerical methods and an appropriate mix of database objects to be stored: Once this estimate is available the second stage of our scheme is directly applicable.

The scheme we propose exploits the knowledge of the hashing function h to reorganize the group assignments in such a manner so that the various g_i, $i = 1, \ldots, G$ have the same average occupancy. We will show the effectiveness of the methodology on both synthetic and real-world databases, and for several hashing functions. The resulting gains in storage utilization and load balancing are significant.

We will describe and discuss our scheme using several hashing functions that have been used extensively with spatial databases of polygonal two-dimensional contours. For the experiments we have used both synthetic workloads, as well as real-world data (database of fingerprints). The obtained experimental results demonstrate that the proposed framework leads to significant benefits when large databases need to be created and maintained.

The paper is organized as follows: Section 2 introduces the data set that will be used throughout this presentation and describes the first stage of our framework. Section 3 discusses the impact of the probability density function of the hash keys on performance, whereas Section 4 describes the second stage of our framework. Several additional hashing functions and Zipf-distributed data properties are introduced in Section 5. Section 6 presents results derived from synthetic data sets and a real-world (fingerprint) database. Related work is discussed in Section 7, while conclusions are found in Section 8.

2 Beginning with a Hashing Function

Let us consider a data set $\mathcal{D}$ comprising D data items d_i, $i = 1, \ldots, D$. Each of the data items is represented by a set of properties and their respective values. These values can be categorical or numerical. Since categorical values can easily be remapped to numerical ones, we will concentrate, without loss of generality, on numerical data only. For the purposes of our discussion, we assume a retrieval system that uses a hash table as its core data structure.

Associated with the hash table is a hashing function h; the function operates on a subset of a data item's properties and generates a hash key that is used to identify and access a hash bucket. The domain of h is either a finite or an infinite set and is typically multidimensional. In the general case, the range of h is a subset of $\mathbb{R}^k$, where k is an integer, i.e., when applied to a data item d_i, the function h will generate one k-dimensional representation (a k-tuple) for d_i; the k-tuple remains invariant or quasi-invariant under a set of "transformations" that d_i can undergo. Often, the desire for resilience to input noise and fault tolerance calls for h to generate a redundant representation of d_i and, thus, more than one k-tuples.

The k-tuples are points in a k-dimensional space of representations. There is at least one point representing each data item and, typically, hashing functions are designed so that "similar" data items have corresponding points that are "close" to each other in the representation space. In essence, a well-behaving hashing function is one that will map *data items* that differ a little to *representations* that also do not differ much. Appropriate metrics, such as Euclidean, Mahalanobis, etc., have been devised to evaluate data-item similarity and quantify it through computation of the distance between the point-representations of the items. It should be mentioned that this is a short, simplified description of the general problem and that a large body of relevant literature exists. The interested reader can refer to [18], [10] for more information.

The raw k-tuples that the hashing function produces cannot be easily used; instead, a form of quantization is

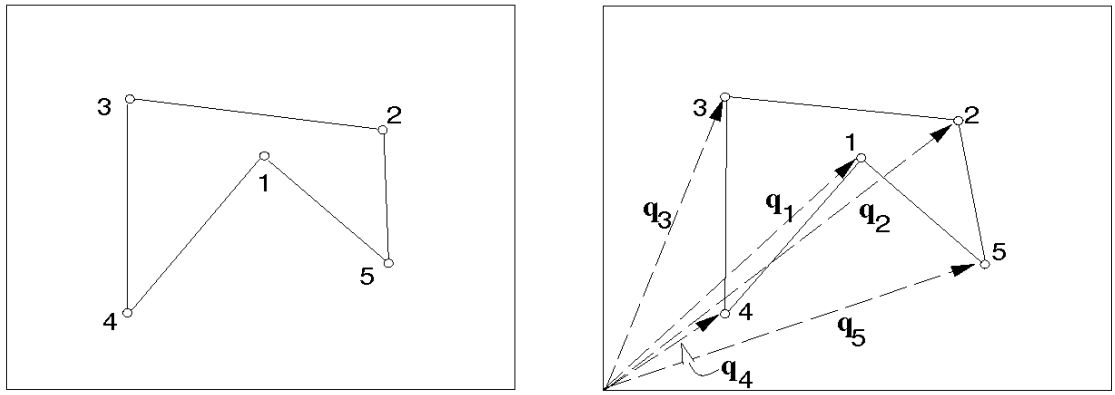


Fig. 1. A model *M* consisting of five points.

typically applied. The *i*th component of all *k*-tuples that the hashing function will generate on the data items d_i assumes a range of values: Linear or nonlinear quantization of this interval into *m* steps allows the remapping of the *i*th component of the tuple to an integer in the interval [0, *m*). Operating similarly on each of the tuple's components (possibly using different quantization schemes for each component) permits one to derive a *k*-tuple of *integer indices* that identifies a hash bucket: In this manner, we can associate data item representations with hash table locations.

One issue that is relevant to this discussion is that of the distributions of the various data item properties in $\mathcal{D}$. Each of the properties is assumed to take values from a (possibly unbounded) domain following a certain distribution. For example, if the data items are community member records, the *i*th property could be a person's *height*. It is known that people's height is a Gaussianly distributed variable; consequently, the values assumed by the *i*th property in this case will follow a Gaussian distribution with a certain mean value (e.g., 5 feet 6 inches) and a certain standard deviation (e.g., 6 inches). Let us denote by *f* the probability density function (pdf for short) that describes the distribution of values that the *i*th property exhibits. We will describe later how this information is used.

In what follows, we will assume that the retrieval system under consideration exposes the knowledge of the hashing function *h* it uses to associate data items with hash buckets. It is also worth noting that when an expression for *h* is not directly available, it can still be approximated using numerical methods.[1] In order to facilitate the understanding of the proposed method, we will define a specific hashing function which we will use to demonstrate the operations needed to implement our scheme. We next describe in more detail the selected hashing function and provide the rationale for this particular choice.

2.1 A Hashing Function on Spatial Data

Let us consider the simple world of two-dimensional polygonal contours, such as the various shapes found in a Chinese tangram set. For simplicity, we treat each such shape as a collection of the respective contour's corner points: Every point has an associated pair of coordinates (*x*, *y*), and the data item is simply a collection of (*x*, *y*) pairs. In Fig. 1, we show such a data item *M* whose contour consists of $n = 5$ points with position vectors $\mathbf{q}_1$, $\mathbf{q}_2$, $\mathbf{q}_3$, $\mathbf{q}_4$, and $\mathbf{q}_5$, respectively.

Consider now any triplet of corner points, $\{\mathbf{q}_i / i = 1, 2, 3\}$ from the contour of a given shape, e.g., points "4," "1," and "3." Observe that the triangle formed by these points will remain unchanged if *M* is rotated or translated (or both) with respect to its original placement. This means that we can actually express the distance of any point $\mathbf{q}$ from the midpoint of "4" and "1" as a linear combination of the unit vectors $(\mathbf{q}_4 - \mathbf{q}_1)/\|\mathbf{q}_4 - \mathbf{q}_1\|$ and $((\mathbf{q}_4 - \mathbf{q}_1)^{\perp}/\|\mathbf{q}_4 - \mathbf{q}_1\|)$.[2] This set of operations is captured more succinctly by the expression:

$$\mathbf{q} - \left(\frac{\mathbf{q}_4 + \mathbf{q}_1}{2}\right) = u\left(\frac{\mathbf{q}_4 - \mathbf{q}_1}{\|\mathbf{q}_4 - \mathbf{q}_1\|}\right) + v\left(\frac{(\mathbf{q}_4 - \mathbf{q}_1)^{\perp}}{\|\mathbf{q}_4 - \mathbf{q}_1\|}\right) \quad (1)$$

and the set of points "4" and "1" is known as the *basis*. In the above equation and for the above selection of points, $\mathbf{q}$ refers to $\mathbf{q}_3$.

It is easy to verify that, because of the rigidity property of the formed triangle, the produced tuples (*u*, *v*) will not change (i.e., will remain *invariant*) under rotation and translation transformations of the model *M*, i.e., if the set of points in *M* is rotated and translated on the plane (Fig. 2), and the above-described procedure is repeated for the new positions of the three selected points, the (*u*, *v*) that will now be generated will be the same as before.

If we fix the first two points of the triplet, i.e., "4" and "1," then as the third point of the triplet assumes the identity of any of the remaining $n - 2$ points of *M*, (1) will generate one (*u*, *v*) pair for each formed triangle. We can repeat the above process for a different selection of the first two points of the triplet (e.g., "2" and "4," instead of "4" and "1"). For each one of the $n(n - 1)$ choices for the first pair there will be $n - 2$ choice for the third point of the triplet giving rise to a total of $n(n - 1)(n - 2)$ pairs (*u*, *v*). The set of these pairs can be used to represent the shape under consideration in a redundant manner.

This particular hashing function, coupled with a redundant representation and used on a database of two-dimensional contours formed the origin of an approach to a model-based object recognition computer system in the mid 1980s.

1. We will return to this issue later in the discussion.

2. The $\perp$ symbol indicates the vector that is perpendicular to its argument, i.e., $\mathbf{q}^{\perp} \cdot \mathbf{q} = 0$. And the operator $\| \ \|$ returns the norm of its operand.

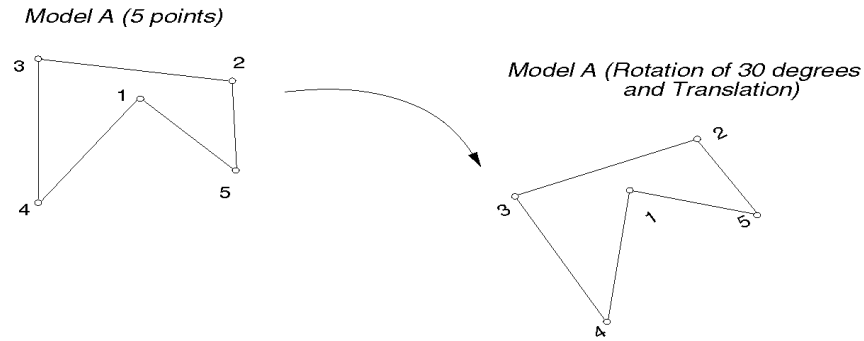


Fig. 2. Rotation and translation of the shape.

The approach is known as *geometric hashing* and since then it has found applications in numerous domains [16], [33], [40]. The one distinguishing characteristic of geometric hashing is that it can quickly generate hypotheses about the identity of the models appearing in a query scene even when objects are partially visible. The method achieves that through a two-phase approach:

1) storage and
2) look-up.

During the storage phase, the hash bucket that can be derived from each quantized pair (u, v) is accessed and an entry containing a description of the model and the basis pair under consideration is made in that bucket. The storage phase takes place off-line and entries are made for all identifiable subsets of point features and for all the models. During the look-up phase, the system is presented with a query that may contain one or more models. Point features are identified in the scene and subsets of those features are used to access the hash table in a manner similar to that of the storage phase. However, instead of making an entry into the accessed bucket, all entries contained therein are retrieved and used to hypothesize the identity and position/orientation of one of the recognizable models in the query scene. A voting step identifies the most likely candidate whose presence in the scene is in turn verified. The reason for the redundant representation of each model by means of multiple entries in the hash table is related to the requirement that a model be recognized even in the presence of occlusion. If a model is partially visible, it is the features that can be identified that will generate the indices with which one could access the hash table. For more information on geometric hashing and reviews of the field, the reader can refer to [27].

It is easy to verify that solving (1) for u and v provides us with solutions that are real numbers and each can assume any value in the interval $(-\infty, +\infty)$. But, the finite size of the used array forces us to restrict our attention to only a finite-size region of the entire $\mathbb{R}^2$ domain. In implementations of hash-based systems, the employed hash tables are typically made to occupy that part of the region of $\mathbb{R}^2$ which accounts for over 90 percent of the total number of entries. Quantization of the u and v values that correspond to the selected region produces a natural mapping to the set of integers: It is these integer values that are used as indices that identify and access hash buckets.

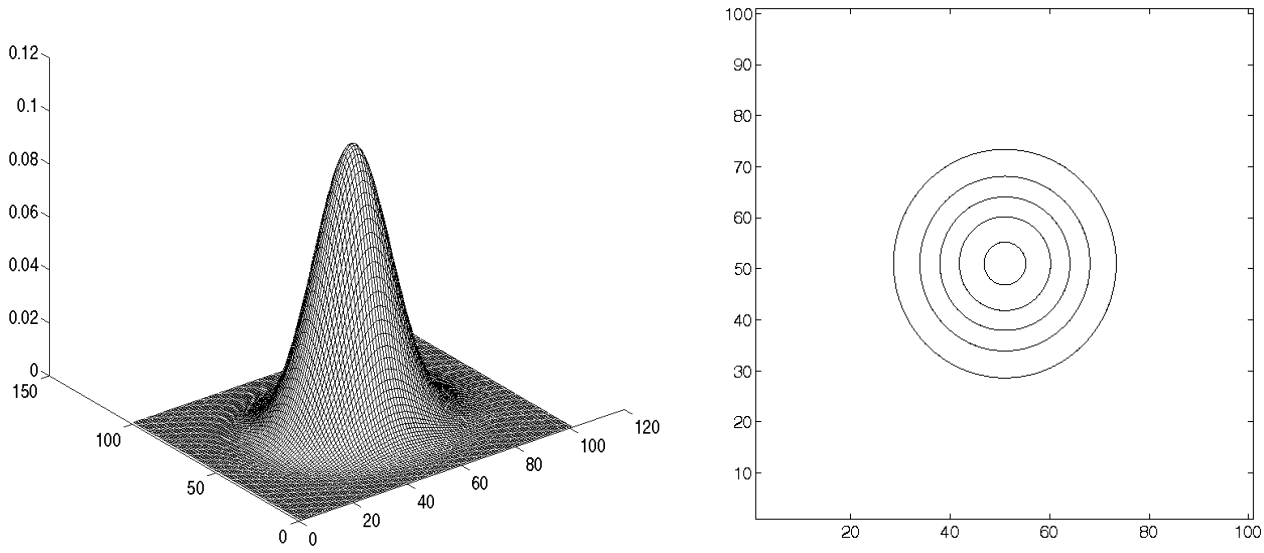
The hashing function that (1) implements is ideal to store and retrieve the objects of our example database of two-dimensional tangram shapes when only rigid transformations (i.e., rotation and translation) are allowed. One can envision an extension to the task of similarity-based retrieval that also allows for the scaling of the tangram shapes. In this case, the shape of Fig. 1 and a scaled-down version of it ought to be considered identical. The above hashing function can easily be modified to accommodate this extension and we will return to this in Section 5. Or, one could also allow for "shear" transformations of the tangram shapes; this is equivalent to saying that the tangram shape of Fig. 1 is identical to the result obtained when a general linear transformation is applied to it. A different modification of the above hashing function allows us to generate retrieval systems that would tackle the problem in this case. As can be seen, the hashing function we will be using as an example with the needed modifications can be used in a gamut of different tasks and will provide a platform for explaining our method.

As stated before, it is necessary to make some assumptions regarding the distribution of values for each of the properties of a data item d_i. Alternatively, one can derive an approximation of these distributions through a histograming of the observed values. For simplicity we will begin with an assumed model for the distribution of values and show in the process how one can use a numerical technique to derive approximations when a model is not available. For the example of the tangram shapes, the observable properties of each data item (i.e., of each shape) is the x and y coordinates of the corner points. Typical choices are Gaussian and uniformly distributed values for the x and y; in the sequel, we will examine various combinations of hashing functions and corner point distributions and will demonstrate different ways to tackle the variations of the problem.

3 The Impact of the Hash Key Density Function

In this section, we will discuss the impact of the hashing function on the hash buckets access patterns of the table that implements it.

Let us use the symbol f to denote the probability density function of the corner points of our tangram models. If the invariant tuples (u, v) and the associated resulting hash keys are produced using (1), it is known from probability



(a) (b)

Fig. 3. Mesh (a) and contours (b) for the Probability Density Function of $f_{UV}(u, v)$ of (3). For visualization purposes, the center has been shifted to (50, 50, 0).

theory that the expected joint–pdf $f_{UV}(u, v)$ for u, and v is given by the expression:

$$f_{UV}(u, v) = \int_{\mathbb{R}^4} f\big(x(u, v), y(u, v)\big) f\big(x_1, y_1\big) f\big(x_2, y_2\big) \left\| J \right\|^{-1} dx_1 dy_1 dx_2 dy_2, \quad (2)$$

where $f(\cdot, \cdot)$ is the pdf of the properties under consideration (in this case the corner point coordinates) and J is the Jacobian of the mapping carried out by the (1).

Let us next assume that the points comprising the tangram shapes of $\mathcal{D}$ are drawn from a two-dimensional Gaussian distribution

$$N\left(0, \begin{pmatrix} \sigma & 0 \\ 0 & \sigma \end{pmatrix}\right)$$

that is centered at (0, 0) and has the same standard deviation σ in both the x and y. The selection of a Gaussian process for modeling the distribution of the corner points is by no means limiting [41], [34]: In fact, in many application domains (e.g., pick-and-place systems, airplane identification systems, personal identification systems, etc.), the Gaussian assumption closely captures the observed distribution of model feature points. Carrying out the substitutions in the previous expressions and computing the four-integral gives us that the expected joint pdf for u and v is:

$$f_{UV}(u, v) = \frac{1}{3\pi\sigma^2} \cdot e^{-\left(u^2+v^2\right)/\left(3\sigma^2\right)}. \quad (3)$$

The range of values for each of the u, v is the interval $[-\infty, \infty]$. As mentioned previously, instead of the entire $\mathbb{R} \times \mathbb{R}$, only a small region (subset of $\mathbb{R} \times \mathbb{R}$) is considered and, through quantization, mapped to the hash table: If a produced invariant falls outside the region considered, no entry is made into the hash table. How one determines the size of the region to map to the hash table is a problem specific task and directly related to the hashing function that is used. Typically, a region $[u_1, u_2] \times [v_1, v_2]$ is selected in such a way that:

$$\int_{u_1}^{u_2} \int_{v_1}^{v_2} f_{UV}(u, v)\, du\, dv \geq 0.9, \quad (4)$$

i.e., one makes sure that a large part of the entire distribution's support is included. In this manner, one can guarantee that the same percentage of the total number of possible hash entries will actually be made in the hash table. It is easy to see that hash functions that give rise to distributions with slow decaying characteristics will necessitate that one map larger regions of the space of invariants to the hash table: Any regular tessellation of the corresponding region will produce a hash table with a few buckets containing a large number of entries, whereas the majority of the buckets will contain a handful of entries, or will be empty.

Returning again to our specific example, we can see that the derived expression for the joint pdf implies that there is a strong preference for generating invariant tuples (u, v) with small values for u, and v. In Fig. 3a, we depict $f_{UV}(u, v)$ as a mesh (the center of the mesh has been shifted to (50, 50, 0) for visualization purposes) for the case where the tangrams' corner points are generated by a Gaussian process and the hashing function used is that of (1).

Fig. 3b shows the contours of the pdf. The height at each mesh point directly expresses the degree of preference for the corresponding value of (u, v). We can also think in terms of the number of entries that will be entered during storage in the hash bucket corresponding to a choice of (u, v): The height at a given mesh point is directly proportional to the number of entries that will exist in the respective bucket. Analogous conclusions can be drawn for the impact of the hashing function on the access patterns of the hash buckets: Those buckets that correspond to mesh points with large height values will be accessed more frequently than the remaining buckets. We conjecture that this observation of an existing bias is endemic to hashing-based retrieval systems [24]. And as we have already mentioned, such biases will translate into:

1) reduced discrimination power as a result of the preference for certain locations of the hash table over others;
2) poor load sharing characteristics [47], [48];
3) uneven I/O patterns; and,
4) poor overall usage of the hash table data structure.

What if the corner points of our tangrams followed a distribution other than the Gaussian that we assumed so far? For example, these points could be uniformly distributed over the unit disc instead, i.e., $f(x, y)$ would be 1 everywhere in the unit disc and 0 anywhere else on $\mathbb{R}^2$. In principle, substitution of this new expression for f in (2) would suffice to derive the expected joint pdf for the values of u and v. Unfortunately, and to the best of our knowledge, an analytical derivation of the expression for f_{UV} in this particular case has not yet been possible. But a different approach can be taken and used whenever derivation of a closed-form expression via analytical methods proves intractable.

In such cases, one resorts to numerical approaches for obtaining approximations f^*_{UV} to the actual expression for f_{UV}. The idea here is that one can guess the functional form of f_{UV}, express it in a parametric manner, and compute the best values of the parameters leading to a best fit to the available data. In essence, this approach bypasses the evaluation of the integral mentioned above and attempts to directly estimate its value directly from the observable data. Here, by "estimating its value" we mean that we derive a functional expression in terms of the variables u and v; this expression is an approximation of f_{UV} but suffices for the purposes of our methodology.

The approximation process begins by accessing the available numerical data that represent a discrete version of the function to be approximated; these data are typically the product of a Monte-Carlo simulation process. The approximate process proceeds with the guessing of the functional form of f_{UV} which is expected to best approximate the data. There is no available "recipe" that prescribes a specific way to proceed with this guessing. Typically, producing combinations by selecting functions from a pool containing standard functions (e.g., log, exp, x, x^2, Gaussian, Cauchy, etc.) suffices. The objective is to determine a suitable functional expression that can be used to describe the available data. The method by which one can arrive at such a determination is data dependent and based on the empirical evaluation of the data. Occasionally, a number of different functional forms with varying degrees of quality of fit may need to be tried before a best such form can be selected.

Note that this methodology can also be used to generate an estimate of the distribution of values of the various properties of the data items d_i under consideration: First, a histogram of those values is produced and, then, a numerical approximation of it derived using the process we just outlined.

Once a functional form has been selected, it is typically of parametric form. Frequently, the expression for f^*_{UV} incorporates the unknown parameters in a nonlinear manner: for example, the expression $f(u, v) = (a(u-b)^2 + c(v-d)^2)^3$ is nonlinear in all of the parameters a, b, c, and d. In situations like this, the values of the parameters need to be computed. This can be achieved by numerical methods such as the *simplex* method [35].

Since the evaluation of the above integral proved intractable when the corner points were uniformly distributed over the unit disc, we applied the outlined recipe in order to derive a numerical approximation of it. It was determined that the function $f^*_{UV}(u, v) = 1/\left(au^2 + bv^2\right)^2$ provided the best approximation and the simplex method helped us determine the values of a, b, and c as

$$f^*_{UV}(u, v) = \frac{1}{\left(\left(4.7u^2 + 3.9v^2\right)^2 + 36.7\right)^2}. \quad (5)$$

We should stress that approximating the probability density function instead of analytically deriving it is of no consequence to the methodology we are suggesting: Indeed, in the following analysis f_{UV} would merely need to be replaced by f^*_{UV} (cf. (5)).

4 The Rest of the Solution Framework

The second and last part of the suggested framework requires that the joint probability density function f_{UV}, or an approximation f^*_{UV} of it be available. As we saw in the previous section, f_{UV} can either be computed from (2) or approximated via numerical methods. To simplify the notation, and without loss of generality, we will overload f_{UV} and use it to denote both f_{UV} and f^*_{UV}.

Once the pdf function $f(u, v)$ is available, the framework that we are suggesting calls for the computation of a transformation that maps the original distribution to the uniform distribution over a closed region (in particular, a rectangle). This new mapping can be seen as a *rehashing* function:

$$g : \mathbb{R}^2 \to \mathbb{R}^2$$

that is used to evenly distribute the hash bucket entries over a rectangular hash table. Notice that the *domain* of the function g is precisely the *range* of function h, i.e., the set of possible invariant tuples (u, v).

The synthesis $(g \circ h)$ is a new hashing function that operates on the data items d_i and generates invariant tuples (u', v') such that the equally spaced hash buckets in this re-mapped space will have a uniform expected population.

For the specific hashing function h that we have been using as an example, let us assume that we have a set of two continuous, stochastic variables U and V. These are the stochastic variables that will assume the values (u, v) produced by the hashing function h. The stochastic-variable-pair members are assumed to be identically and independently distributed with a joint probability density function f_{UV}. We define two functions *New_u*( ) and *New_v*( ) on these variables as:

$$New\_u(U, V) : \mathbb{R}^2 \to \mathbb{R}$$

and

$$New\_v(U, V) : \mathbb{R}^2 \to \mathbb{R}.$$

These two functions are assumed to have continuous partial derivatives of the first order in all of $\mathbb{R}^2$ and define a mapping $\mathbf{T} : (U, V) \in \mathbb{R}^2 \rightarrow \mathbb{R}^2 \ni (New\_U, New\_V)$. Then, the joint pdf f_{New_U,New_V} of the stochastic variables $New\_U = New\_u(\,)$ and $New\_V = New\_v(\,)$ is given by:

$$f_{New\_U,New\_V}(New\_u, New\_v) = \int_{\mathbb{R}^2} f_{UV}\big(U(New\_u, New\_v), V(New\_u, New\_v)\big) \big|J\big|^{-1} dU\, dV, \quad (6)$$

where J is the Jacobian of the transformation $\mathbf{T}$; given the above assumptions, the Jacobian is defined all over $\mathbb{R}^2$ and is $\neq 0$. The objective of the framework's second stage is to compute the transformation (mapping) $\mathbf{T}$ so that the resulting joint pdf f_{New_U,New_V} is uniform over a closed region. These mappings are typically case-dependent. Occasionally, more than one mappings may produce the desired property. As a rule of thumb, the following mapping can frequently be computed easily and has the desired properties:[3]

$$\mathbf{T}(u_0, v_0) = \left(\int_{-\infty}^{u_0} \int_{-\infty}^{\infty} f(u, v)\, du\, dv, \int_{-\infty}^{\infty} \int_{-\infty}^{v_0} f(u, v)\, du\, dv \right). \quad (7)$$

As already stated, other mappings will occasionally be possible. We will demonstrate this statement in the context of the hashing function of (1). In order to simplify the computation of the mapping that is suggested above, we first remap the Cartesian coordinates to polar ones. The coordinates in the Cartesian domain are nothing but the invariants u and v that are produced by the original hashing function. If ρ and θ denote the polar coordinates, then by definition we have that $\rho = \sqrt{u^2 + v^2}$ and $\theta = \arctan(u, v)$. Rewriting (3) in polar coordinates, we have that:

$$f(\rho, \theta) = \frac{1}{3\pi\sigma^2} \cdot e^{-\rho^2/(3\sigma^2)}. \quad (8)$$

Notice that although the range of values for u and v was the interval $[-\infty, \infty]$, for the polar coordinates ρ and θ the respective ranges are $[0, \infty)$ and $[0, 2\pi)$, and this needs to be taken into account when deriving the remapping $\mathbf{T}$. Notice that:

$$\int_0^{\rho_0} \int_0^{2\pi} f(\rho, \theta)\, d\theta\, \rho d\rho = \left(1 - e^{-\rho_0^2/(3\sigma^2)} \right) \quad (9)$$

and that:

$$\int_0^{\infty} \int_0^{\theta_0} f(\rho, \theta)\, d\theta\, \rho d\rho = \theta_0, \quad (10)$$

we can see that the remapping $\mathbf{T}$ in this case will be equal to:

$$\mathbf{T}(u, v) = \big(h_1(u, v), h_2(u, v)\big) = \left(1 - e^{-(u^2+v^2)/(3\sigma^2)}, \frac{\big(\text{atan2}(v, u) + \pi\big)}{2\pi} \right), \quad (11)$$

where

1) we have used the function atan2 instead of arctan because the former returns the phase in the interval $[-\pi, \pi]$, and
2) we have divided the atan2 component by 2π so that both components assume values between 0 and 1, and the remapped invariants will be uniformly distributed over the region $[0, 1] \times [0, 1]$ of the space of invariants.

We conclude this section by noting that it suffices to use as a hashing function the synthesis $g \circ h$, where the functions g and h have been derived above. Indeed, given the properties of the component functions g and h, it is clear that the synthesis $g \circ h$ is a function that will map a set of feature points to an invariant tuple whose distribution will exhibit the above listed desired properties.

The above procedure can of course be repeated in the case of corner points that are assumed to be uniformly distributed over the unit disc: In this case, only an approximation for f_{UV} is available. In this case, the remapping $\mathbf{T}$ will be equal to:

$$\mathbf{T}(u, v) = \big(h_1(u, v), h_2(u, v)\big) = \left(\frac{2}{\pi} \text{atan}\left(\frac{(au)^2 + (bv)^2}{c^2} \right) + \frac{2\Big[(au)^2 + (bv)^2\Big]c^2}{\pi\left(\Big[(au)^2 + (bv)^2\Big]^2 + c^4\right)}, \big(\text{atan2}(v, u) + \pi\big)/(2\pi) \right). \quad (12)$$

5 Other Hashing Functions and Property Distributions

The outline of the two steps of the suggested framework uses the hashing function h of (1). As already mentioned, this hashing function is appropriate whenever the tangram shapes of our database can undergo rigid transformations, i.e., rotations and translations. We will now extend the problem of matching the tangram shapes when the more general similarity and affine transformations are possible. In this manner, we will demonstrate the generality of the framework through application to other hashing functions. We will also consider Zipf-distributed properties of data items and derive the respective formulas.

When the tangram shapes can undergo scaling transformations in addition to rotation and translation, the hashing function of (1) needs to be modified slightly. The new equation is

$$\mathbf{q} - \left(\frac{\mathbf{q}_4 + \mathbf{q}_1}{2} \right) = u\big(\mathbf{q}_4 - \mathbf{q}_1\big) + v\big(\mathbf{q}_4 - \mathbf{q}_1\big)^{\perp}. \quad (13)$$

In this expression, we used the notation associated with Fig. 1. Equation (13) expresses the distance between the midpoint of "4" and "1" and any of the remaining points of M as linear combination of the vectors $(\mathbf{q}_4 - \mathbf{q}_1)$ and $((\mathbf{q}_4 - \mathbf{q}_1)^{\perp}$. It is easy to verify that solving this equation for u and v provides two numbers that remain invariant under similarity transformations of the tangram shape under consideration.

Finally, the tangram shapes may be allowed to undergo an affine transformation, the most general transformation on the plane that also incorporates "shear" in addition to rotation, translation and scaling. The relevant expression in this case is different from the ones we have already presented. The

3. This operation is in essence histogram equalization [34], and has been used in many contexts [1].

TABLE 1
TWO MORE COMBINATIONS OF HASHING FUNCTION AND FEATURE DISTRIBUTIONS

Transformation / Distribution	PDF of Hash Key Distribution	Remapping Function
Case : Similarity / Gaussian	$f(u,v)=\frac{12}{\pi\left(4\left(u^2+v^2\right)+3\right)^2}$	$h_1(u,v)=1-\frac{3}{4\left(u^2+v^2\right)+3}$ $h_2(u,v)=(\text{atan2}(v,u)+\pi)/(2\pi)$
Case : Affine / Gaussian	$f(u,v)=\frac{2\sqrt{2}}{\pi\left(4u^2+4v^2+4uv+8/3\right)^{\frac{3}{2}}}$	$h_1(u,v)=1-\frac{2\sqrt{2}}{\sqrt{3}\sqrt{(4u^2+4uv+4v^2+8/3)}}$ $h_2(u,v)=\left(\text{atan2}\left(u\sqrt{3}+v\sqrt{3},u-v\right)+\pi\right)/(2\pi)$

most important difference has to do with the fact that three (not two) points are needed now to form a basis:

$$\mathbf{q}-\left(\frac{\mathbf{q}_4+\mathbf{q}_2+\mathbf{q}_1}{3}\right)=u\left(\mathbf{q}_4-\mathbf{q}_1\right)+v\left(\mathbf{q}_2-\mathbf{q}_1\right). \quad (14)$$

Notice also that it is the distance from the barycenter of the triangle formed by the three points participating in the formation of the basis that is now expressed as a linear combination of two distinct vectors.

Of course, and as already stated, there is also the issue of the distribution of properties (in this case corner-point positions) of our data items, a consideration that is orthogonal to the choice of the hashing function.

We have repeated the above outlined steps of the framework for several combinations of hashing functions and property distributions. Additionally, we have examined the case where the properties follow a Zipf law. We treat the Zipf-law distributed properties as a special case and discuss it separately below; the remaining two cases are shown in Table 1. For each of the two combinations shown there, we list the expression for the joint pdf of the invariants u and v together with the function $\mathbf{T} = (h_1(u, v), h_2(u, v))$ that will remap (u, v) to (u', v') so that the latter are uniformly distributed over a closed region of the space of invariants.

Property distributions frequently encountered in database applications follow Zipf's law, Z_k [49], [20]. To further validate our methodology, we generated synthetic tangram shapes whose corner-points followed a two-dimensional Zipf distribution with constant k; the tangram shapes were allowed to undergo a similarity transformation. We considered three distinct cases of the Zipf distribution, those corresponding to $k = 0.1$, $k = 0.2$, and $k = 0.5$, and repeated our methodology.

Derivation of an expression for the joint pdf by evaluation of the integral in (2) proved impossible. Consequently, a numerical approach was followed. We discovered that a good functional form to approximate all three distinct joint pdfs was given by:

$$f(u,v)=\frac{A}{\left(B\,u^2+C\,v^2+D\right)^2} \quad (15)$$

with the values of A, B, C, and D being dependent on the value of k used. In fact, using the simplex method we determined the values of the unknown constants in each of the three cases $k = 0.1$, $k = 0.2$, and $k = 0.5$.

Given this last expression and following the outlined methodology, it is possible to verify that the appropriate remapping equations are:

$$\mathbf{T}=\left(h_1(u,v),h_2(u,v)\right)=\left(\frac{1}{2}\cdot\left(1+\sqrt{\frac{B}{D+Bu^2}}u\right),\frac{1}{2}\cdot\left(1+\sqrt{\frac{C}{D+Cv^2}}v\right)\right). \quad (16)$$

This remapping can be used in all three cases, after the appropriate values have been substituted for B, C, and D.

But there is also an alternative remapping that one could use. This second remapping entails first translating the Cartesian coordinates (u, v) into polar (ρ, θ) and then applying our outlined methodology. For this particular expression, the way to carry out the translation is by making use of the mapping:

$$\rho=\sqrt{\left(B\,u^2+C\,v^2\right)^2}$$
$$\theta=\text{atan2}\left(\sqrt{C}v,\sqrt{B}u\right).$$

Once the remapping to polar coordinates has been carried out, it is straightforward to verify that an alternative remapping, $\mathbf{T}_{alter}$ would be:

$$\mathbf{T}_{alter}=\left(h_1(u,v),h_2(u,v)\right)=\left(1.0-\frac{D}{D+Bu^2+Cv^2},\frac{\left(\text{atan2}\left(\sqrt{C}v,\sqrt{B}u\right)+\pi\right)}{2\pi}\right). \quad (17)$$

As before, this remapping can be used in all three cases, after the appropriate values have been substituted for B, C, and D.

Table 2 shows the values for A, B, C, and D, as these have been determined by the simplex method for each of the three Zipf distributions that we examined. In the experimental results section, we will show graphically the results and quality of this remapping for $k = 0.2$.

6 EXPERIMENTAL RESULTS

In this section, we first show results that demonstrate how even a very simple hashing functions can lead to a nonuniform distribution of generated invariants; we also demonstrate how the quality of a hashing function can be misjudged if the used hash table does not contain a sufficiently large number of entries. We then present the results derived from both synthetic and real data and discuss the merits of our framework.

6.1 Even Simple Hashing Functions May Create Problems

Let us assume a data set consisting of 100,000 entries describing one-dimensional line segments. These segments

TABLE 2
THE VALUES OF THE PARAMETERS IN THE APPROXIMATING EXPRESSION FOR VARIOUS CHOICES OF THE ZIPF'S CONSTANT *k*

Distribution	K	A	B	C	D
Zipf-1	0.1	0.1680	0.4037	0.3403	1.3085
Zipf-2	0.2	0.1658	0.3935	0.3556	1.2939
Zipf-3	0.5	0.1610	0.3768	0.4307	1.2328

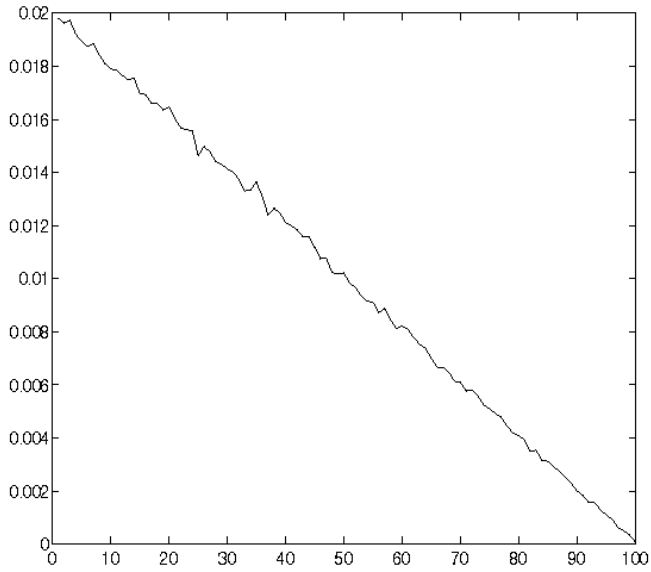


Fig. 4. The pdf for the length of the line segments shown discretized over a set of 100 intervals (x-axis).

are to be recognized independent of any translation transformations they may be subject to. The vertices of each line segment are selected uniformly from the unit interval [–0.5, 0, 5]. One way to represent such segments is by means of their length. Since all lengths will assume values in the interval [0, 1], we can quantize this interval with *B* many buckets and use a line segment's length to decide which bucket to store any relevant information (i.e., the line segment's identity, etc.). A hashing function for this example would be:

$$h(x_i, x_j) = |x_i - x_j|_q,$$

where x_i and x_j are the coordinates of the line segment under consideration, and *q* indicates that the absolute value has been "quantized." Despite the fact that the vertices of each line segment are drawn with uniform probability from the unit interval, the occupancy density over the entire set of buckets shows preference for some values from the range of possible values of *h* over others. This is shown in Fig. 4, which depicts the discretized pdf for *h*. Apparently, shorter segments are more populous, resulting into a highly skewed pdf curve.

In a more complex setting, the data set contains 100,000 two-dimensional triangular shapes. These shapes are to be recognised independent of any rotation, translation, and scaling they may have undergone; in other words, if a triangle Δ is one of the triangles in the set, *M* is a rotation matrix, *T* a translation matrix, and *s* a scaling factor, then when presented with the triangle:

$$\Delta' = M(s\Delta) + T,$$

the system should identify it as being identical to Δ by retrieving all relevant information about Δ from the data set. For simplicity purposes, we assume that the vertices of all the *model* triangles that will ever be stored in the data set are drawn with uniform probability from the unit square [–0.5, 0.5] × [–0.5, 0.5]. Let us denote by $\left(x_{i_j}, y_{i_j}\right)$, j = 1, 2, 3, the three vertices of the *i*th model triangle that is in the data set. One hashing function that can "encode" model triangles in a manner that is rotation, translation, and scaling independent is the following:

$$h\left(x_{i_1}, y_{i_1}, x_{i_2}, y_{i_2}, x_{i_3}, y_{i_3}\right) = \left(\alpha_{i_1}, \alpha_{i_2}\right)_q,$$

where α_{i_1} and α_{i_2} denote the angles at the vertices 1 and 2, of the *i*th triangle in the data set, expressed in radians, and *q* denotes that the values of the two-tuple of invariants have been "quantized." Notice that since:

$$\alpha_{i_1} + \alpha_{i_2} + \alpha_{i_3} = \pi,$$

inclusion of the angle α_{i_3} in the hashing function is redundant. Once again, the occupancy density over the entire set of buckets shows preference for several values from the range of possible values of $h(\cdot, \cdot, \cdot, \cdot, \cdot, \cdot)$ over others. And this is despite the fact that the vertices of each triangle Δ are drawn with uniform probability from the unit square. Fig. 5 shows the mesh and contours of the discretized joint probability density function for this second hashing function. Similar to Fig. 4, the obtained pdf shows a highly skewed behavior that certainly leads to uneven load distribution and diverse storage requirements for buckets.

6.2 Misjudging the Quality of a Hashing Function

At times, it is possible that the quality of the used hash functions can be misjudged. The goal of this experiment is to show this fact. A 30 × 30 bucket hash table is used to accommodate 1,200, 12,000, 120,000, and 1,200,000 entries in four different settings. The allowed transformation is rigid and the points were assumed to be Gaussianly distributed. The used hashing function (3) was kept the same during the experiments but only the number of hash entries increased by an order of magnitude each time. We have generated the meshes for the occupancies of the hash table buckets resulting from these four different settings. The meshes obtained for the 1,200 and 12,000 populations resulted in visually flat-looking meshes. However, the meshes and contours for 120,000, and 1,200,000 entries in the hash table buckets (shown in Fig. 6) reveal significant statistical behavior. The scaling of the axes is the same in order to show the appearance of statistical behavior as the number of entries increased. For hash functions of different dimensionality, the number of stored entries required before this statistical behavior reveals itself is expected to be different. Nevertheless, the continuous increase of the population will ultimately reveal statistical behavior.

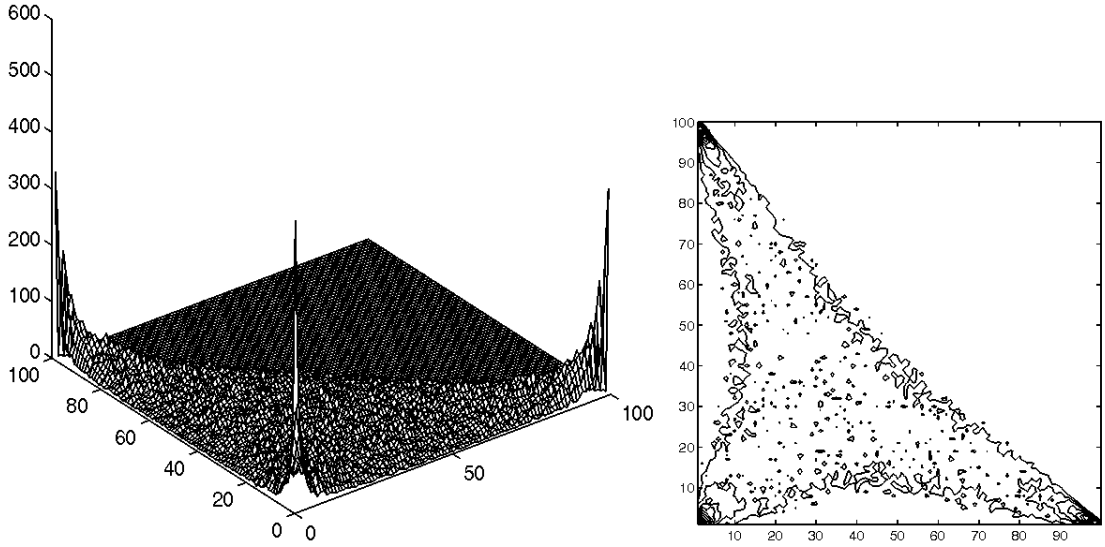


Fig. 5. Mesh and contours for the pdf for the angles of a triangle; the vertices of the formed triangles were uniformly distributed within the unit square. The x- and y-axis have been divided into 100 distinct intervals and the shown extent corresponds to one π exactly.

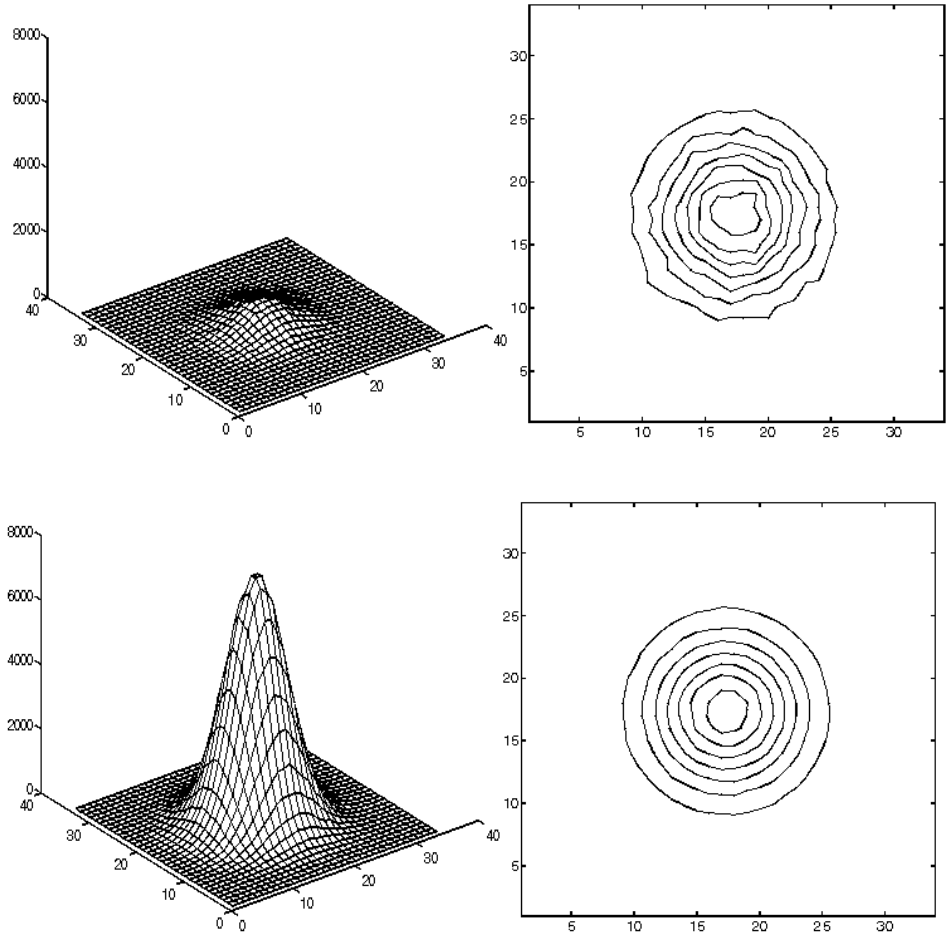
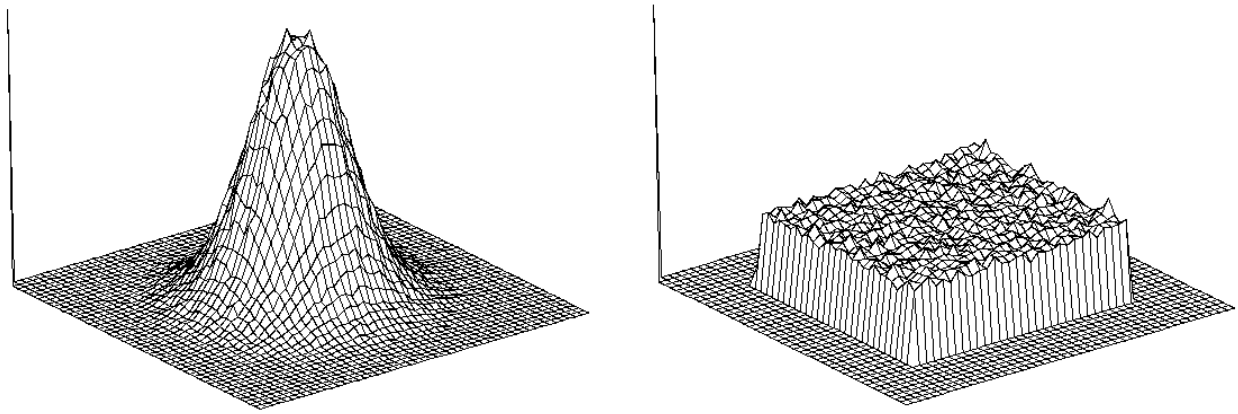


Fig. 6. Mesh and contours for the occupancy of the hash table when 120,000 and 1,200,000 entries, respectively, are used. The employed hashing function is the one of (3).

6.3 The Efficiency of the Suggested Framework

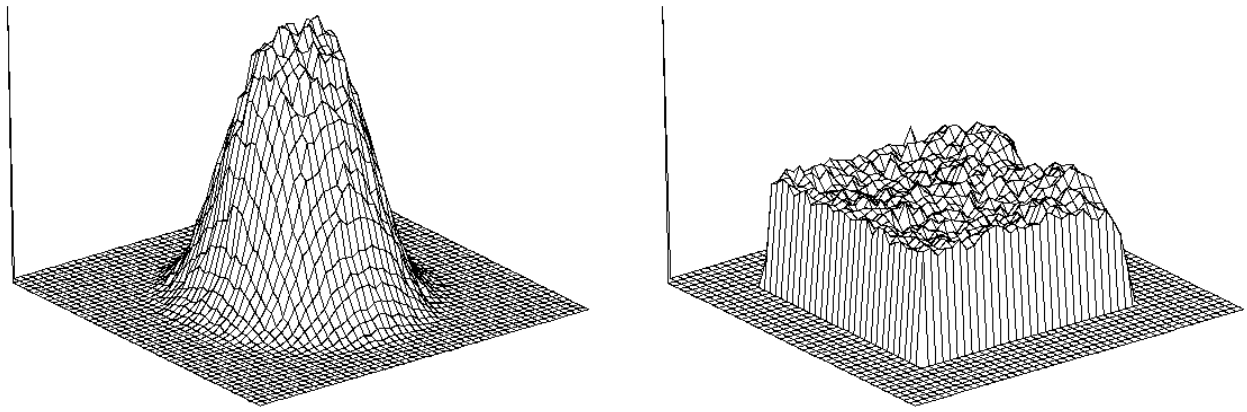
So far, we have focused on the used hashing function and assumed properties for the distribution of the coordinates of the feature points comprising a model. We now present results from experiments that use both a synthetically generated and a real-world database.

In particular, we carried out a Monte-Carlo simulation to synthetically derive tangram-like shapes such as those used



(a) (b)

Fig. 7. The synthesized corner-points follow a Gaussian distribution. (a) Mesh for the pdf corresponding to the hashing function of (1). (b) Mesh for the pdf when the appropriate remapping function is used.



(a) (b)

Fig. 8. The synthesized corner-points are uniformly distributed over the unit disc. (a) Mesh for the pdf corresponding to the hashing function of (1). (b) Mesh for the pdf when the appropriate remapping function is used.

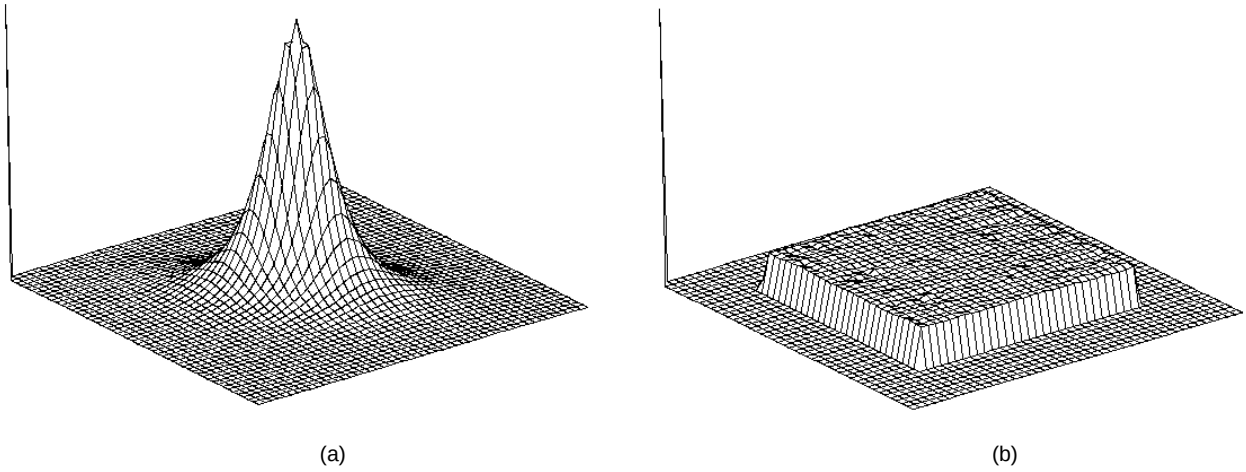
in our discussion. The corner points of these shapes were drawn

1) from a two-dimensional Gaussian distribution and
2) from a uniform over the unit disc distribution.

We used the corner point coordinates and the shown hashing functions to make entries for each object in a two-dimensional hash table. For each entry, we determined the identity of the receiving bucket by applying the synthesis $g \circ h$ to the coordinates of the triplet of vertices that gave rise to the entry in question, We kept track of two hash tables: One was the table that resulted from use of the hashing function h, whereas the second was the table we obtained for $g \circ h$. Both hash tables comprised an equal number of hash buckets; the fact that both tables had the same number of buckets facilitated the evaluation of the method. Indeed, the hash entries were distributed over the same size hash tables but using the two hashing functions h and $g \circ h$. A ring of buckets in the periphery of the table for $g \circ h$ was intentionally left unused in order to improve the visualization of the results.

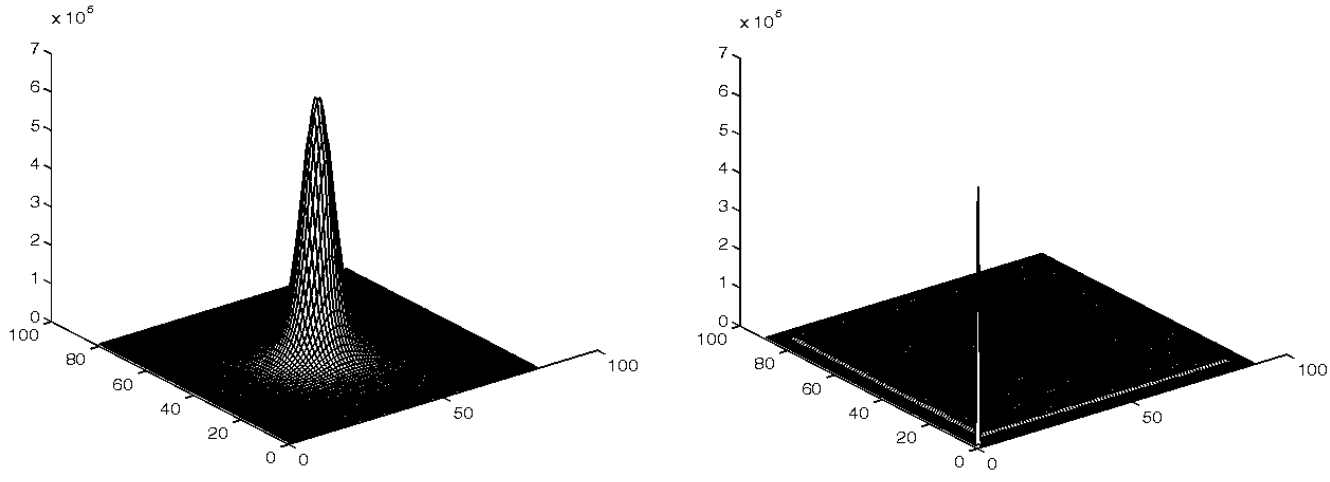
The height of the mesh at a given hash bucket location is proportional to the number of entries in the bucket. There is a total of 10 million entries stored in each of the two hash tables. Notice that each of the meshes has a "reference spike" in one of the corners: The spike's height is equal to the number of entries of the *fullest* bucket in the hash table corresponding to h and provides a measure of the benefits obtained when $g \circ h$ is used. These results indicate that the occupancy behavior for both h and $g \circ h$ is in full agreement with the predictions of our analysis above. It is worthwhile noting that the shown pdf's which were derived after applying the h functions contain at least 97.5 percent of the generated hash entries; recall that the hash table can only accommodate a finite region of the entire space of invariants and, thus, not all of the generated entries will land inside the hash table. Naturally, when the synthesis of functions is used, *all* of the generated hash entries landed in the hash table.

Fig. 7 shows the meshes for the occupancies of the buckets when the tangrams' corner points followed a Gaussian distribution and the hashing function was that of (1). Fig. 8 shows the respective two meshes when the tangrams' corner points are distributed uniformly over the unit disc; the hashing function is again that of (1). In Fig. 9, we show the



(a) (b)

Fig. 9. The synthesized corner-points are Gaussian distributed. (a) Mesh for the pdf corresponding to the hashing function of Case 2 in Table 1. (b) Mesh for the pdf when the respective remapping function is used.



(a) (b)

Fig. 10. (a) Mesh for the pdf corresponding to the similarity transformation, with the features following a Zipf distribution with $k = 0.2$. (b) Mesh for the pdf mesh corresponding to $g \circ h$.

mesh for the occupancies of the buckets for the hash tables corresponding to Case 2 of Table 1.

Next, we present the results we obtained from our Monte-Carlo simulations using Zipf-distributed properties. In particular, the x and y coordinates of the tangrams' corner points were drawn independently from the Zipf distribution that corresponded to k value of 0.2. The formulae from the previous section were used to produce the hash tables for h and $g \circ h$. The tangram shapes were allowed to undergo similarity transformations. Fig. 10 shows the meshes for the occupancies of the buckets for the hash tables corresponding to the hashing function of (13) (similarity transformation) and the resulting flat-like $g \circ h$ in the case where $k = 0.2$. The results corresponding to the cases $k = 0.1$ and $k = 0.5$ were identical in flavor. Hence, the respective graphs are omitted for brevity.

Finally, we show results obtained using a real-world database. The elements to be recognized were fingerprints of 600 individuals. Each fingerprint was represented as a collection of minutiae, i.e., points of interest corresponding to the locations of things like bifurcations, swirls, ridge terminations, etc. The minutiae were used to derived the models ultimately stored in the hash structure. The only information available was the positional information about a fingerprint's minutiae. In that respect, the models were not any different than the tangrams used in our discussion. The allowed transformation was 2D-affine and the feature points were distributed uniformly over the image square (512 pixels $\times$ 512 pixels).

What makes this case different is the fact that when the features are uniformly distributed over a convex domain (i.e., square, disc, ellipse, etc.), the affine-invariant hashing function of (14) exhibits a few unusual and unexpected properties. In a companion report [38], we study an alternative affine hashing function and how it behaves when the feature domain is convex and unspecified. Fig. 11 shows the meshes for both the hashing function and the corresponding $g \circ h$ transformation. Notice that unlike the functions that we have examined so far, this affine-invariant hashing function exhibits a *discontinuity*. Due to the fact that both

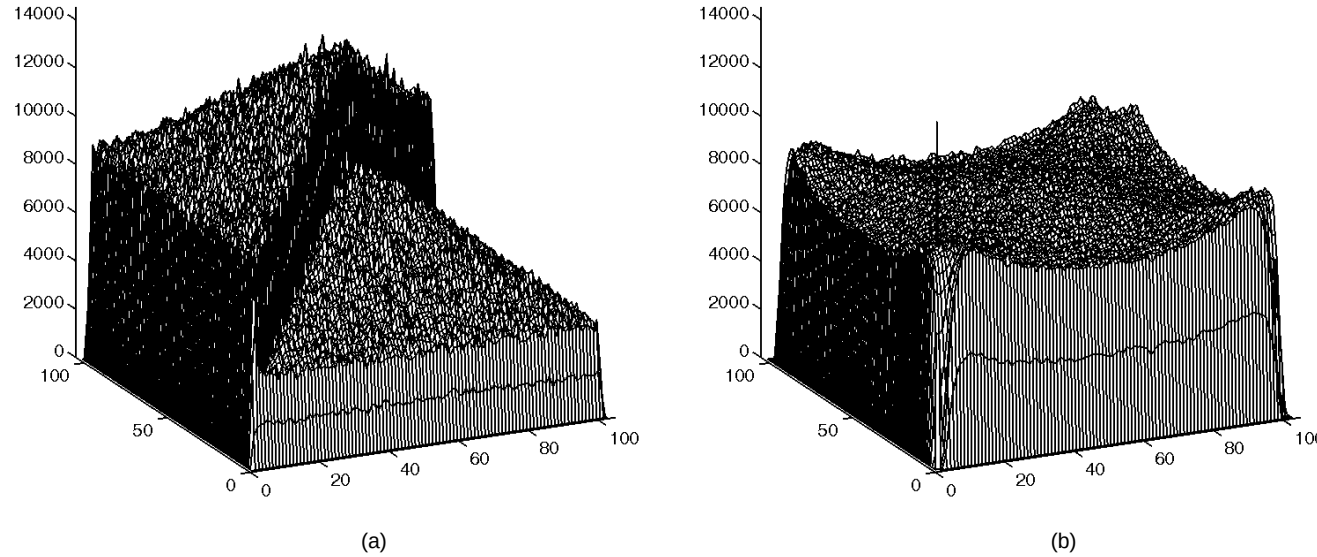


Fig. 11. (a) Mesh for the pdf corresponding to the affine-invariant hashing function mentioned in the text. The database used contains fingerprints from 600 individuals and the features under consideration are the fingerprint *minutiae* together with their positional information. The features were uniformly distributed over the image square (512 pixels $\times$ 512 pixels). (b) Mesh for the pdf mesh corresponding to $g \circ h$.

the description of the used hashing function as well as of the remapping $g \circ h$ are fairly involved, details are omitted; the interested reader can refer to [38]. It should be stressed, however, that it was the methodology outlined in this discussion that was used in that case and resulted in the quasi-flat pdf of Fig. 11.

7 Related Work

Work related to our effort falls into three categories: hashing, multidimensional indexing techniques, and distribution of access structures.

An extensive survey of hashing functions and strategies for handling overflow in buckets is discussed in [24]. In [36], a number of hashing functions are tested and their performance verified against data of very modest sizes. Bentley and Friedman [4] provide a definition of the range-search problem on multiple attributes. Multidimensional data (known also as k-d data) are accessed and processed by representations in lower dimensions. A number of approaches take the view that k-d data can be transformed in one-dimensional representations and, thus, traditional access methods can be used for their manipulation. Gutman's R–trees and variations [17], [43], [3] use this idea. The hB-tree [42] is based on the k-d tree and splits are done using several attributes, unlike in R-trees, where splits are done using a hyper-plane. The tree is not balanced and can become severely skewed, thus affecting search time. In [12], Gray coding is used as the technique for the mapping to one–dimension. Peano and Hilbert curves have also been used in the same context [21]. The usual assumptions of uniformity and independence are contended in [13]. The concept of the fractal dimension is proposed as the means to describe the deviation from uniformity. The Grid File [32] follows an approach orthogonal to ours, where the data space is split and merged depending on the population of the buckets. Guenther and Buchmann [15] present a taxonomy of spatial access methods and highlight some of their deficiencies.

As the volume of databases may exceed the available size of storage devices on a single machine, their data have to be handled by an environment that provides multiple disk units and possibly processors. Early work on distributed structures can be found in [11]. Bastani et al. [2] address the issue of reliability in distributed indexing. In [23], a structure called the "Multiplexed R-tree" is proposed; this is essentially a single R-tree with pointers spanning disks. Honishi et al. [19] proposed a method where the index structure is partitioned into subtrees indexed by a hash structure. Matsliach and Shmueli [30] presented a parallel access method suitable for shared-memory multiprocessor systems. Johnson et al. [22] propose the use of the dB-tree, which is based on the B-link tree [28], as a solution suitable for parallel striped file systems. This is done by storing ranges of index keys in an abstraction called an extent, which is indexed by a B-link tree. By tightly integrating the conceptual network topology and the search structure to be distributed, Kroll and Widmayer [26] simplify the distribution of nodes among processors.

In many instances, deterioration of the performance happens due to uneven load distribution [7], [44], [29], [25], [47]. In the context of our work, our objective has been to have the various data records evenly distributed among a set of available buckets. In this manner, we can guarantee an even partitioning of the incurred load among the available processors (or workstations) by equally distributing the hash table buckets. Desirable scalability properties result as the probability density function $f_{g \circ h}$ of the invariants is flat-like over the entire *finite* region of the new produced values. This implies that for a given regular discretization of the interval of produced invariant values, the preference for an entry ending in a given hash table bucket is, on the average, constant and independent of the bucket under consideration. Furthermore, such an occupancy behavior implies that the employed invariants provide us with the maximum possible degree of discrimination among the stored objects.

8 Conclusions

Very large data sets expose the statistical properties of hashing functions as they create nonuniform bucket occupancy distributions in the hash tables. In this paper, we present a two-stage methodology used to remap data sets so that the resulting hashing scheme exhibit competitive performance characteristics. We showcase it with an actual hashing function for a spatial database and derive expressions for the distribution of bucket occupancy over the space of possible indices via a parametric estimation step, or, whenever possible, through an analytical determination of a closed form expression. The derived distribution is then exploited using methods from probability theory to *redesign* the hashing function so that exhibits flat-looking occupancy properties for its hash table buckets.

The presented framework helps us to achieve desirable scalability features. Given a fixed size hash table, increasing the number of stored entries by a factor k will result in a performance slowdown by the same factor: The employed hashing scheme guarantees that the bucket occupancy, although k times as high now as before, will continue being constant *on the average* across the set of hash table buckets. The increase in the load and storage requirements will impact all of the available processors (or workstations) equally. Similar arguments apply to the case where the amount of available processing power increases by a factor k: There will be a corresponding decrease in the load and storage requirements at every processor (or workstation) resulting in a system that exhibits a k-fold improvement in performance.

While there is a wealth of hashing techniques and distributed / parallel access methods, none of these techniques deals with the retrieval of objects based solely on partial information. These methods rely on explicit keys such as unique keywords and spatial characteristics. More importantly, none deals with the probabilistic behavior of hashing indices. By introducing our framework and the stochastic treatment of the bucket occupancy patterns, our aim is to fulfill the following major requirements:

- Storage of a very large number of data using multiple key "features" in hash table access structures. This large number reveals highly skewed occupancy patterns.
- Easy application of our framework in a parallel and multicomputer architectures to solve realistic problems offering very short response times.
- Exploitation of the resulting occupancy distributions in order to generate hash access tables with close-to-uniform occupancy patterns.
- Ability to query data objects based on partial information.

The framework is also essentially immune to "large-scale" insertions and deletions of objects as it is capable of maintaining similar utilization rates throughout all the used hashing buckets.

Although we have demonstrated the approach in the context of a specific example, the methodology should be treated as a general framework that allows us to predict and subsequently improve the performance of hashing-based retrieval systems. Furthermore, the dimensionality of the hashing function h is by no means restricted to two. Indeed, any dimension can be assumed for h; the suggested methodology will remain directly applicable.

Acknowledgments

The authors would like to thank Sharath Pankanti for kindly providing the fingerprint database that was used in the experiments, and the reviewers for their comments. Alex Delis was supported in part by the U.S. National Science Foundation under grant NSF IIS-9733642.

References

[1] D. Ballard and C. Brown. *Computer Vision*. Prentice Hall, 1982.
[2] F. Bastani, S. Iyengar, and I. Yen, "Concurrent Maintenance of Data Structures in a Distributed Environment," *The Computer J.*, vol. 31, no. 12, pp. 165-174, 1988.
[3] N. Beckmann, H.P. Kriegel, R. Schneider, and B. Seeger, "The R*-Tree: An Efficient and Robust Access Method for Points and Rectangles," *Proc. ACM SIGMOD Conf.*, Atlantic City, N.J., 1990.
[4] J.L. Bentley and J.H. Friedman, "Data Structures for Range Searching," *ACM Computing Surveys*, vol. 11, no. 4, pp. 397-409, 1979.
[5] A. Califano and R. Mohan, "Multidimensional Indexing for Recognizing Visual Shapes," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 16, no. 4, pp. 373-392, 1994.
[6] J.L. Carter and M. Wegman, "Universal Classes of Hash Functions," *J. Computer and System Sciences*, vol. 18, no. 2, 1979.
[7] T. Casavant and J. Kuhl, "A Taxonomy of Scheduling in General Purpose Distributed Computing Systems," *IEEE Trans. Software Eng.*, vol. 14, no. 2, Feb. 1988.
[8] M. Costa, R. Haralick, and L. Shapiro, "Optimal Affine-Invariant Matching: Performance Characterization," *Proc. SPIE Conf. Image Storage and Retrieval*, San Jose, Calif., Jan. 1992.
[9] R.F. Deutscher, P.G. Sorenson, and J.P. Tremblay, "Distribution-Dependent Hashing Functions and Their Characteristics," *Proc. SIGMOD Conf.*, pp. 224-236, San Jose, Calif., 1975.
[10] R. Duda and P. Hart, *Pattern Classification and Scene Analysis*. New York: John Wiley and Sons, 1973.
[11] C. Ellis, "Distributed Data Structures: A Case Study," *IEEE Trans. Computers*, vol. 34, no. 12, pp. 1,178-1,185, Dec. 1985.
[12] C. Faloutsos, "Gray Codes for Partial Match and Range Queries," *IEEE Trans. Software Eng.*, vol. 14, no. 10, pp. 1,381-1,393, Oct. 1987.
[13] C. Faloutsos and I. Kamel, "Beyond Uniformity and Independence: Analysis of R-trees Using the Concept of Fractal Dimension," *Proc. 1994 ACM PODS Conf. Principles of Database Systems*, Minneapolis, Minn., 1994.
[14] K.K. Goswami, M. Devarakonda, and R.K. Iyer, "Prediction-Based Dynamic Load-Sharing Heuristics," *IEEE Trans. Parallel and Distributed Systems*, vol. 4, no. 6, pp. 638-648, June 1993.
[15] O. Guenther and A. Buchmann, "Research Issues in Spatial Databases," *ACM SIGMOD Record*, vol. 19, no. 4, pp. 61-67,1990.
[16] A. Guéziec and N. Ayache, "Smoothing and Matching of 3D Space Curves," *Proc. European Conf. Computer Vision*, Santa Margherita Ligure, Italy, May 1992.
[17] A. Guttman, "R-Trees: A Dynamic Index Structure for Spatial Searching," *Proc. 1984 SIGMOD Conf.*, pp. 47-57, 1984.
[18] J. Hartigan, *Clustering Algorithms*. Wiley and Sons, 1975.
[19] T. Honishi, T. Satoh, and U. Inoue, "An Index Structure for Parallel Database Processing," *IEEE Second Int'l Workshop Research Issues on Data Eng.*, pp. 224-225, 1992.
[20] Y. Ioannidis, "Universality of Serial Histograms," *Proc. 19th VLDB Conf.*, Dublin, Ireland, 1993.
[21] H.V. Jagadish, "Linear Clustering of Objects with Multiple Attributes," *Proc. 1990 ACMSIGMOD Conf. Management of Data*, Atlantic City, N.J., 1990.
[22] T. Johnson, P. Krishna, and A. Colbrook, "Distributed Indices for Accessing Distributed Data," *Proc. 12th IEEE Symp. Mass Storage Systems*, pp. 199-207, 1993.
[23] I. Kamel and C. Faloutsos, "Parallel R-Trees," *Proc. ACM SIGMOD Conf.*, pp. 195-204,1992.
[24] G. Knott, "Hashing Functions," *The Computer J.*, vol. 18, no. 3, Mar. 1975.

[25] O. Kremien and J. Kramer, "Methodical Analysis of Adaptive Load Sharing Algorithms," *IEEE Trans. Parallel and Distributed Systems*, vol. 3, no. 6, Nov. 1992.
[26] B. Kroll and P. Widmayer, "Distributing a Search Structure Among a Growing Number of Processors," *Proc. ACM SIGMOD Conf.*, pp. 265-276, 1994.
[27] Y. Lamdan and H. Wolfson, "Geometric Hashing: A General and Efficient Model-Based Recognition Scheme," *Proc. Int'l Conf. Computer Vision*, pp. 238-249, 1988.
[28] P. Lehman and S. Yao, "Efficient Locking for Concurrent Operations on B-Trees," *ACM Trans. Database Systems*, vol. 6, no. 4, pp. 650-670, 1981.
[29] M. Livny and M. Melman, "Load Balancing in Homogeneous Broadcast Distributed Systems," *ACM Performance Evaluation Review*, vol. 11, no. 1, Spring 1982.
[30] G. Matsliach and O. Shmueli, "An Efficient Method for Distributing Search Structures," *Proc. First Int'l Conf. Parallel and Distributed Information Systems*, pp. 159-166, 1991.
[31] R. Mohan, D. Weinshall, and R. Sarukkai, "3D Object Recognition by Indexing Structural Invariants from Multiple Views," *Proc. Int'l Conf. Computer Vision*, pp. 264-268, Berlin, May 1993.
[32] J. Nievergelt, H. Hinterberger, and K.C. Sevcik, "The Grid File: An Adaptable, Symmetric Multikey File Structure," *ACM Trans. Database Systems*, vol. 9, no. 1, pp. 38-71, 1984.
[33] R. Nussinov and H.J. Wolfson, "Efficient Detection of Three-Dimensional Motifs in Biological Macromolecules by Computer Vision Techniques," *Proc. Nat'l Academy Science USA*, vol. 88, pp. 10,495-10,499, 1991.
[34] A. Papoulis, *Probability, Random Variables, and Stochastic Processes*. New York: Mc Graw-Hill, 1984.
[35] W.H. Press, B. Flannery, S.A. Teukolsky, and W.T. Vetterling, *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge Univ. Press, 1988.
[36] M.V. Ramakrishna, "Hashing in Practice, Analysis of Hashing and Universal Hashing," *Proc. 1988 ACM-SIGMOD Conf. Management of Data*, Chicago, 1988.
[37] M.V. Ramakrishna, E. Fu, and E. Bahcekapili, "Efficient Hardware Hashing Functions for High Performance Computers," *IEEE Trans. Computers*, vol. 46, no. 12, pp. 1,378-1,381, Dec. 1997.
[38] I. Rigoutsos, "Well-Behaved, Tunable 3D Affine-Invariants," *Proc. IEEE Computer Vision and Pattern Recognition Conf.*, Santa Barbara, Calif., June 1998.
[39] I. Rigoutsos and A. Califano, "Searching in Parallel for Similar Strings," *IEEE Computational Science and Eng.*, pp. 60-75, Summer 1994.
[40] I. Rigoutsos, D. Platt, A. Califano, and B.D. Silverman, "Representation and Matching of Small Flexible Molecules in Large Databases of 3D-Molecular Information," *Pattern Discovery in Molecular Biology*, J. Wang, B. Shapiro, and D. Shasha, eds. Oxford Univ. Press, to appear.
[41] R. Sahner, K. Trivedi, and A. Puliafito, *Performance and Reliability Analysis*. The Netherlands: Kluwer Academic Publishers, 1995.
[42] B. Salzberg, "Practical Spatial Database Access Methods," *IEEE Symp. Applied Computing*, pp. 82-90, 1991.
[43] T. Sellis, N. Roussopoulos, and C. Faloutsos, "The R+-Tree: A Dynamic Index for Multidimensional Objects," *Proc. 13th Very Large Data Bases Conf.*, pp. 507-518, 1987.
[44] A. Svensson, "History, and Intelligent Load Sharing Filter," *Proc. 10th Int'l Conf. Distributed Computing Systems*, 1991.
[45] X. Wang, J. Wang, D. Shasha, B. Shapiro, S. Dikshitulu, I. Rigoutsos, and K. Zhang, "Automated Discovery of Active Motifs in Three Dimensional Molecules," *Proc. Third Int'l Conf. Knowledge Discovery and Data Mining* (*KDD '97*), Newport Beach, Calif., Aug. 1997.
[46] G. Wiederhold, *Database Design, Computer Science Series*, second ed. New York: McGraw Hill, 1983.
[47] M.H. Willebeek-LeMair and A.P. Reeves, "Strategies for Dynamic Load Balancing on Highly Parallel Computers," *IEEE Trans. Parallel and Distributed Systems*, vol. 4, no. 9, pp. 979-993, Sept. 1993.
[48] S. Zhou, M. Stumm, K. Li, and D. Wortman, "Heterogeneous Distributed Shared Memory," *IEEE Trans. Parallel and Distributed Systems*, vol. 3, no. 5, pp. 540-554, Sept. 1992.
[49] G.K. Zipf, *Human Behavior and the Principle of Least Effort*. Reading, Mass.: Addison-Wesley, 1949.

Isidore Rigoutsos received his BSc in physics from the University of Athens, Greece, and his PhD in computer science from the Courant Institute of Mathematical Sciences of New York University. Since 1992, he has been with IBM's T.J. Watson Research Center, where he is currently the manager of the Bioinformatics and Pattern Discovery Group. His research activities focus on computational biology, computer vision, applied mathematics, and parallel computing. Dr. Rigoutsos is a member of the AAAS, the ACM, and the IEEE.

Alex Delis received his PhD in computer science from the University of Maryland at College Park and his Diploma in computer engineering from the University of Patras, Greece. He is an assistant professor in the Department of Computer and Information Science at Polytechnic University in Brooklyn, New York. His research interests are in distributed databases and software engineering. His work has been supported by the U.S. National Science Foundation through a Career Award, the Australian Research Council, New York State, the UTRC, and SIAC Corporation. He is a member of the IEEE, the ACM, Sigma Xi, and the New York Academy of Sciences.