RC 20648 (12/05/1996)
Large Databases

IBM Research Report

Managing Statistical Behavior In Large Data Sets

Isidore Rigoutsos

IBM Research Division
T.J. Watson Research Center
Yorktown Heights, New York

Alex Delis

Computer and Information Science
Polytechnic University
Brooklyn, NY 11201

IBM Research Division
Almaden · T.J. Watson · Tokyo · Zurich

Managing Statistical Behavior In Large Data Sets

Isidore Rigoutsos
IBM T.J. Watson Research Center,
PO BOX 704,
Yorktown Heights, NY 10598

Alex Delis
Computer and Information Science
Polytechnic University
Brooklyn, NY 11201

December 10, 1996

Abstract

Very large data sets make the statistical properties of hashing functions become apparent. Such properties are manifested by the utilization patterns of buckets. Non-uniform bucket occupancy distributions have been experimentally established in databases that support identification, biological, and spatial systems. In this paper, we discuss the impact that the choice of a hashing function has on load balancing in very large data sets. We propose a two-stage methodology that allows us to exploit the features of a suggested hashing function. We exploit these features to re-design hashing functions so that they exhibit a number of desired properties. Experimental results from synthetic as well as real data are presented. The discussed methodology can be generally applicable to a large number of storage and/or retrieval systems and results in significant performance improvements.

1 Introduction

Although conventional hashing techniques perform sufficiently well with databases of small sizes (tens of thousands of data records), it is becoming increasingly evident that one must be cautious when applying such techniques to very large data sets [25, 3, 29]. Very large data sets expose the statistical properties of hashing functions. Such properties are manifested by the patterns of utilization demonstrated by the hash table buckets. Non–uniform occupancy of distributed buckets leads to ineffective load–balancing and increased query response time. A number of hashing functions, that could potentially be used with databases handling spatial applications, exhibit non-uniform occupancy distributions. This has been observed with databases of biological information and one-dimensional strings of amino acids and nucleic acids [4]: although the employed hashing functions are eight-dimensional ones, as the number of items in the database increased, non-uniform occupancies developed. Gray-encoding techniques for modifying the hashing function can alleviate the problem to an extent but a certain degree of non-uniformity still persists. It should be noted here that previous use of Gray-encoding as described in [9, 17] does not address the issue of modifying the hashing function but rather the way the multi-dimensional array can be treated as a linear space and traversed accordingly; as such, the method cannot guarantee optimal performance in a manner independent of the pattern of accessing the database's records. In addition, the method by which items are stored in and retrieved from this linear space is based on traditional access methods.

In applications that involve model-based object recognition, geometric hashing has been recently adopted [6, 11, 12]. Objects are handled as groups of points in space, termed models. Each object is ultimately represented by one such model. In the geometric hashing technique, a two-dimensional and continuous hashing function is being used. This hashing function generates invariants that are exploited in an intelligent manner to store and later retrieve two- and or three-dimensional objects from a database of known objects. Invariants are "tuples" of values (of arity m) derived from object models and remain constant even if models are subject to a number of transformations (such as scaling, translation, rotation etc.). The m values that constitute these m-ary tuples are also known as indices since they may uniquely identify positions in a m-dimensional space. One of the main characteristics of geometric hashing is the non-uniform distribution of index values over the space of the generated invariants [6, 25]. We conjecture that this non-uniformity is not specific to a particular data type, but instead is endemic to all access methods that are built around a hashing scheme. To this point, it has been encountered in a large variety of domain/application combinations [3, 37, 29].

In general, this observed non-uniformity raises a number of interesting and important questions. From a theoretical standpoint, it is important to know the range of values that the generated indices can assume. Related to this, the frequency of appearance of an index allows one to draw conclusions about the discriminating power the index carries. Also, it is conceivable that certain hashing functions may give rise to multi-modal distributions in the space of invariants; it is thus important to be able to determine the location(s) of the hash buckets with the largest expected number of entries, as well as the occupancy pattern in the neighborhood of these buckets. Finally, as it has been shown in previous work, the knowledge of the distribution of indices over the space of invariants is *instrumental* to the development of Bayesian approaches to database access in the presence of noisy data inputs [27].

From a practical standpoint, the non-uniform distribution over the space of invariants results in different lengths for the hash entry lists. Since the length of the longest such list impacts on the time needed to process the data during a query, a *non*-uniform 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 data structure. Additionally, in a parallel implementation of the method, an almost constant occupancy of all the hash buckets results in an improved load balancing among the processors. Knowledge of the expressions that describe the distributions of the invariant values over the space of invariants greatly facilitates the equalization of the hash bucket occupancy. To this point, and to the best of our knowledge, there exists very little published work on determining quantitatively and/or qualitatively the index distributions that one can anticipate. In [35], a stochastic approach was used for clustering database objects and it was shown that optimal clustering is an NP-complete problem.

We can identify three different approaches in the existing bibliography for alleviating the problem of non-uniform distribution. First, one can increase the dimensionality of the hashing function [3, 25]. Second, use of database-specific heuristics can improve the occupancy pattern of the hash table buckets [6]. Finally, a third possibility is the one we are advocating in this paper: one can make implicit or explicit use of *irregular* tessellations of the space of indices: traditionally, the hash table used in a store/retrieve system has been obtained by a regular tessellation of a region of the space of invariants; i.e. the widths of any two buckets of the hash table were equal. In an irregular tessellation though, this property will not in general be there and the locations of the boundaries of the buckets will in general be given by a non-linear expression. Let us assume that we have a one-dimensional function $f(x)$ that can assume values between 0 and a, $a \in N$. An example of an *irregular* tessellation of the interval $[0, a]$ is given by the function $pos(i) = \frac{i}{a+1-i}$, where $pos(i)$ $(i \geq 0)$, gives the position of right boundary of the i-th bucket; notice that this tessellation will divide the interval under consideration into a many buckets. As we will discuss below, it is frequently the case that such tessellations prove beneficial if discrimination ability, optimal resource usage and speed are of concern.

In our approach, we begin with the knowledge of the actual hashing function that the store/retrieve system under consideration 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.

During the first stage the probability density function for the indices computed by the hashing scheme is determined: this can be done either via a parametric estimation step, or, whenever possible, through the computation of a closed form expression from first principles and the knowledge of the hashing function. Once the probability density function for the invariants generated by the hashing function is available, well-defined methods from probability theory can be used to re-design the hashing function so that it now exhibits a set of desired properties. For the purposes of using a specific example, we will present our scheme within the context of a spatial database of polygonal two-dimensional contours. However, it should be stressed that the method is very general and, in principle, is applicable every time that the hashing function used by a store/retrieve system is known to the user.

We demonstrate the validity of our approach using both synthetic workloads and real-world data. The obtained experimental results demonstrate that the proposed framework can have significant contributions in the way very large databases can be created and maintained. But what is more important, our experiments also lead to the conclusion that the proposed framework can be successfully used to improve the actual performance (both from a speed and discrimination ability standpoint) of database systems which handle very large amounts of data. Our approach easily extends to the case of an environment where the "buckets" are collections of disk pages.

The paper is organized as follows: Section 2 discusses the main features of geometric hashing and Section 3 describes the means that allow us to describe the statistical behavior of the resulting invariant entries of the indices. Section 4 outlines our approach in re-shuffling the data points obtained in the pure geometric hashing phase so that "flat–looking" distributions are obtained. Diverse transformations and the handling of Zipf-distribution functions are discussed in Section 5. Section 6 presents results derived from synthetic and real data sets. Related work is discussed in Section 7, while conclusions are found in Section 8.

2 Brief Introduction To Geometric Hashing

In this section, we describe in detail the geometric hashing approach for accessing spatial databases of two-dimensional objects. We present and develop our methodology in this specific context for the simple reason that this approach has dominated the field of object recognition systems for a number of years and has given rise to numerous, successful working systems. The databases that geometric hashing operates on are spatial databases of two-dimensional or three-dimensional objects. It should be stressed again that our choice to work in this context does not limits the validity of our methodology which could be used in a number of other areas [36].

Physical objects are often represented by collections of points that are known as feature points. A typical choice for those points is curvature extrema and/or discontinuities along the contours of the recognizable objects. In [1] the reader can find a good description of methods that begin with a gray-level image of the recognizable objects and generate a set of respective feature points; the methods can operate on both two-dimensional and three-dimensional objects. For the purposes of our discussion, we will assume that these sets have already been described for the groups of objects under consideration.

The main idea behind geometric hashing is the redundant representation of subparts of a recognizable model with the help of a set of numbers which remain invariant when the object under consideration undergoes one of the allowed transformations. The association between parts of a model and the invariant numbers is carried out by the hash table: the buckets of the hash table are accessed by means of the generated invariants. As a result of combining the use of a hash table with the redundant representation, the geometric hashing method allows one to quickly generate hypotheses about the identity of the models appearing in a query scene even when the objects are occluded (i.e. partially visible).

Let us assume that a *subset* of a model's feature-point set generates an invariant number (index) i. During the storage phase, the hash bucket that corresponds to the index i is accessed and an entry containing a description of the model and of the feature subset under consideration is made in that bucket. The storage phase takes place off-line and entries are made for all identifiable subsets of features and for all the models.

During the recognition phase, the system is presented with a query that may contain one or more models. 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 stage 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 occluded, then, only part of it is visible and it is the features that can be identified in this part that will generate the indices with which to access the hash table. It is worth mentioning that if an algorithm (such as geometric hashing) can perform well in the presence of occlusion, then it can also produce answers, i.e. hypothesize model identities and placements, even when the set of recognizable models can only *partially* match the query scene object(s).

Let us for simplicity assume that all of our models are collection of points. These collections of points would otherwise be the feature points that a feature detection mechanism would identify and associate with a model. Let us furthermore assume that each such model is allowed to undergo a rigid transformation on the plane, i.e. some combination of a 2D-rotation and a 2D-translation.

In general, two points are needed to define a Cartesian system C of coordinates on the plane. In Figure 1, we show a model M consisting of five points whose position vectors are $\mathbf{q}_1, \mathbf{q}_2, \mathbf{q}_3, \mathbf{q}_4$ and $\mathbf{q}_5$ respectively.

We begin by selecting an *ordered* pair of model points, e.g. "4" and "1." For each of the remaining points of the set M we express the distance of this point from the midpoint of "4" and "1" as a linear combination of the unit vectors $(\mathbf{q}_4 - \mathbf{q}_1)/ \parallel \mathbf{q}_4 - \mathbf{q}_1 \parallel$ and $((\mathbf{q}_4 - \mathbf{q}_1)^{\perp}/ \parallel \mathbf{q}_4 - \mathbf{q}_1 \parallel)$. This set of operations is captured more succinctly by the expression:

$$\mathbf{q} - (\frac{\mathbf{q}_4 + \mathbf{q}_1}{2}) = u(\frac{(\mathbf{q}_4 - \mathbf{q}_1)}{\parallel \mathbf{q}_4 - \mathbf{q}_1 \parallel}) + v(\frac{(\mathbf{q}_4 - \mathbf{q}_1)^{\perp}}{\parallel \mathbf{q}_4 - \mathbf{q}_1 \parallel}) \tag{1}$$

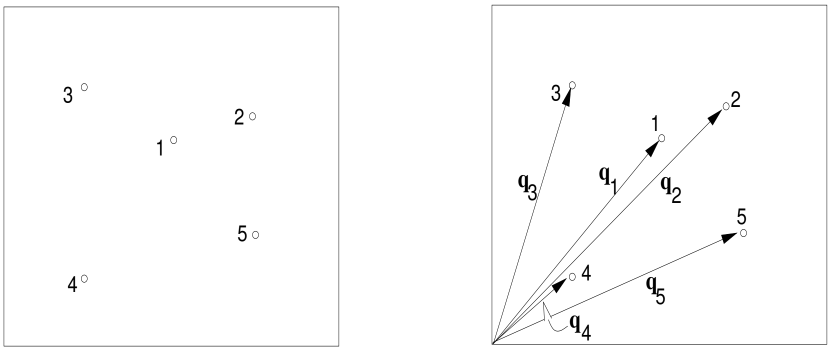


Figure 1: A model M consisting of five points.

where $\mathbf{q}$ will in turn be each of the remaining $\mathbf{q}_2, \mathbf{q}_3, \mathbf{q}_5$.

For each selection for q we will obtain one set of values for the tuple (u, v). It is straightforward to verify that the produced tuples (u, v) are 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 (Figure 2), and the above-described procedure is repeated for the new positions of the points, the tuples that will be generated by $\mathbf{q}_2, \mathbf{q}_3, \mathbf{q}_5$ will be identical to those generated from the original placing of M.

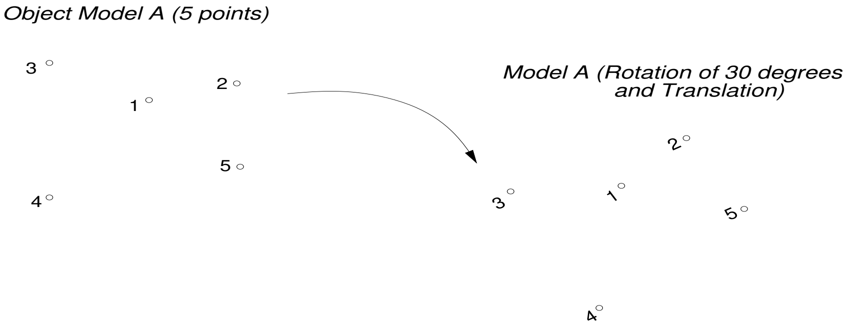


Figure 2: Rotation and Translation of the Points of the Model

This last expression (Eqn. 1) in essence associates (maps) a set of three planar points (i.e. $\mathbf{q}_4, \mathbf{q}_1$ and $\mathbf{q}$) with a set of two numbers (i.e. u, v); this mapping will remain the same even after the set of three planar points has undergone a rigid transformation. It should be evident that this expression in essence implements a hashing function. The procedure is repeated for all possible *ordered* pairs of points from M, and for all models M. If the model M contains N many points, the procedure will produce a hash table which will contain $(N-2)$ entries of the form $(M, (4, 1))$, $(N-2)$ entries of the form $(M, (4, 2))$, etc.

Solving the above equation for u and v provides us with solutions that are real numbers and each can assume any value in the interval $[-\infty, +\infty]$. In order to be able to use these numbers as indices into the n-dimensional array that implements the hash table, these numbers need to be quantized and mapped to integers. Moreover, the finite size of the used array immediately forces us to restrict our attention to only a finite-size region of the entire $\Re^2$ domain.

In implementations of hash-based systems the employed hash tables are typically made to occupy that part of the region of $\Re^2$ which accounts for 90+% of the total number of entries. Quantization of the u and v values which correspond to the selected region produces a natural mapping to the set of integers: it is these integer values that are used as indices into a hash table bucket. It is likely that some hash table buckets will contain more than one entry: indeed, recall that a model with N points will generate $\binom{N}{3}$ entries in the hash table. Typically, the number M of models stored in the hash table is such that $M\binom{N}{3} > B$, where B is the number of hash table buckets. The observation follows as a result of the pigeonhole principle. Consequently, the final hash table data structure will contain a list of entries of the form $(model, basis)$ attached to each hash table bucket.

During the retrieval phase, we are given a query set comprising N' points that belong to one of the database models and which has undergone rotation and/or translation. We select two points, $(\mathbf{p}_{\mu_1}, \mathbf{p}_{\mu_2})$ from the query set. The coordinates of each of the remaining $(N'-2)$ points, $\mathbf{p}$, are then calculated from the expression:

$$\mathbf{p} - \left(\frac{\mathbf{p}_{\mu_1} + \mathbf{p}_{\mu_2}}{2}\right) = u'\left(\frac{(\mathbf{p}_{\mu_1} - \mathbf{p}_{\mu_2})}{\| \mathbf{p}_{\mu_1} - \mathbf{p}_{\mu_2} \|}\right) + v'\left(\frac{(\mathbf{p}_{\mu_1} - \mathbf{p}_{\mu_2})^{\perp}}{\| \mathbf{p}_{\mu_1} - \mathbf{p}_{\mu_2} \|}\right) \tag{2}$$

We basically apply the hashing function $h()$ to the set of three points $\mathbf{p}_{\mu_1}, \mathbf{p}_{\mu_2}$ and $\mathbf{p}$. The computed values for u' and v' are again quantized and the corresponding hash table bucket accessed. All the entries in the accessed hash table bucket receive a vote. If there are sufficient votes for one or more $(model, basis)$ combinations, then a subsequent stage may attempt to verify the presence of a model with the designated *basis* matching the chosen pair $(\mathbf{p}_{\mu_1}, \mathbf{p}_{\mu_2})$.

Finally, let us consider the situation where a model M is not present in the query scene in its entirety but instead only K feature points of it are present. In the context of the above equation (Eqn. 2), the method will generate $\binom{K}{3}$ hits to the hash table and will recover an equal number of entries that pertain to the model M and are consistent in terms of the placement of M that they dictate. However due to the fact that the accessed buckets contain multiple entries, it is likely that other models may get some support just by pure chance. Successful recognition of M is still possible, if and only if the K features that remain visible generate support for M that will exceed any threshold that should be imposed to prevent accidental matches. A description of the algorithm in the presence of "noisy" inputs can be found in [21].

3 The Impact Of The Probability Density Function

In this section, we will examine the impact that the probability density function of the generated invariants has on a storage/retrieval system's characteristics; the hashing function is assumed determined and fixed. We will continue using the hashing function of Eqn. 1.

Up to this point, we made no mention about the distribution of the feature points that are used to form the feature subsets on which the hashing function will operate. Let us next assume that the points comprising the planar sets (i.e. recognizable models) of database D are drawn from the following two-dimensional Gaussian distribution: $N\left(0, \begin{pmatrix} \sigma & 0 \\ 0 & \sigma \end{pmatrix}\right)$.

In other words, the Gaussian process is assumed centered at $(0,0)$ and having the same standard deviation σ in both of its dimensions. Our selection of a Gaussian process for modeling the distribution of feature points is by no means limiting. As matter of fact, in many application domains (e.g. pick-and-place systems, airplane identification systems, personal identification systems etc.) the Gaussian distribution assumption closely captures the observed distribution of model feature points, and it is by no means restrictive. In other domains, the assumption of model points that are uniformly distributed over a square can be more appropriate [27].

If the invariant tuples (u, v) are produced using the Eqn. 1, and under the assumption of Gaussianly distributed feature points, probability theory dictates that the expected joint probability density function (= joint pdf) $f_{UV}(u, v)$ for u, and v is given by the expression:

$$f_{UV}(u,v) = \int_{\Re^4} f\left(x(u,v), y(u,v)\right)\ f(x_1, y_1)\ f(x_2, y_2)\ \|J\|^{-1}\, dx_1\, dy_1\, dx_2\, dy_2 \tag{3}$$

where $f(\cdot,\cdot)$ is the Gaussian distribution shown above and J is the Jacobian of the mapping carried out by Eqn. 1. Doing the substitutions in the previous expressions and computing the four-integral gives us that the joint pdf for u and v is

$$f_{UV}(u,v) = \frac{1}{3\pi\sigma^2} \cdot e^{-\left(\frac{u^2+v^2}{3\sigma^2}\right)}. \tag{4}$$

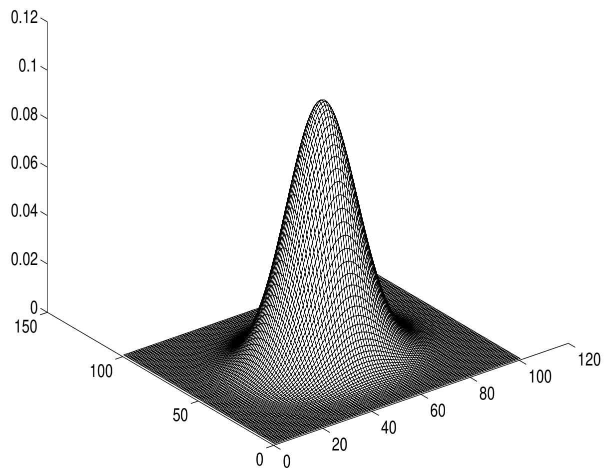
The range of values each of the u, v can assume is the interval $[-\infty, \infty]$. As mentioned previously, instead of the entire $\Re \times \Re$ only a small region (subset of $\Re \times \Re$) is considered and mapped to the hash table: if a produced invariant falls outside the region considered, no entry is made into the hash table.

How one determines the size of the region to map to the hash table is directly related to the type of transformation that the models can undergo. Put differently, the size of the region is dictated by the hashing function that is used. Typically, the region will be equal to $[u_1, u_2] \times [v_1, v_2]$ and

$$\int_{u_1}^{u_2} \int_{v_1}^{v_2} f_{UV}(u, v) \; du \, dv \; >= 0.9 \tag{5}$$

i.e. one makes sure that a large part of the entire distribution's support is included. In this manner, one guarantees that a large portion of the total number of possible hash table entries will actually be made into the hash table. It is easy to see that hash functions that give rise to distributions with slow decaying characteristics will necessitate that one move a long distance from the $(0, 0)$ in order to incorporate as much of the distribution as possible. This in turn necessitates that the hash table have a large extent on $\Re^2$: 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 Figure 3(a), we depict $f_{UV}(u, v)$ as a mesh for the case where the points are generated by a Gaussian process and the allowed transformation is rigid. Figure 3(b) shows the contours of the probability density function. The height at each mesh point directly expresses the degree of preference for the corresponding value of (u, v). But we can also think in terms of the number of entries that will be entered during storage in the bucket of the overlaid hash table that corresponds to a value of (u, v): the height at a given mesh point is directly proportional to the cardinality of the set of $(model, basis)$ entries to be found in the respective bucket.



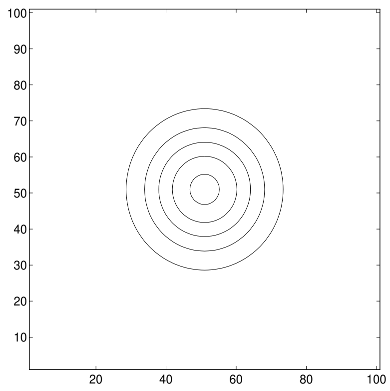


Figure 3: (a) Mesh for the Probability Distribution Function (b) Contours for the *pdf*

Summarizing, we observe that for the hashing function under consideration, there is a strong bias for certain hash table buckets. We conjecture that such biases are endemic to all hashing-based methods for database retrieval, and as we have already mentioned, these 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;
3. poor I/O behavior; and,
4. poor use of the hash table data structure.

All these are issues that store and retrieve systems need to address in order to maximize the benefits induced by a hashing scheme, and/or multiple processors (or workstations).

Before concluding this section, we would like to point out the following: in the analysis above, it was possible to evaluate the multiple integral and determine its value. However, it is also possible that determining a closed-form expression for f_{UV} via analytical methods may prove intractable. In such cases, one can always resort to numerical approaches for obtaining approximations to the actual expression for f^*_{UV} of f_{UV}.

The idea here is to guess the functional form of f_{UV}, express it in a parametric manner and attempt to 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, which in this case is a functional expression in terms of the variables u and v. As we will see in the next section, having the approximate expression f^*_{UV} of f_{UV} 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. The objective here 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 most of the time it is based on the empirical evaluation of the data. For all practical purposes, this step is an art form in itself. Occasionally, a number of different functional forms with varying degrees of quality of fit may have to be tried before a best such form can be selected.

Once a best functional form is chosen, it is typically in a parametric form. Again, most of the time the expression for f^*_{UV} incorporates the unknown parameters in a nonlinear manner: for example, the expression $f(u,v) = \left(a(u-b)^2 + c(v-d)^2\right)^3$ is nonlinear in all of the parameters a, b, c, d, and e. In situations like this, the values of the parameters can typically be computed via a nonlinear minimization approach.

Nonlinear minimization techniques suffer from their own set of shortcomings. Examples of these shortcomings include:

- getting trapped in local minima; the minimization procedure will end up missing better solutions that may exist.
- lack of convergence; the procedure may find itself in a multidimensional valley and end-up oscillating between the ridges of the valley.
- sensitivity to the starting values; the parameters of the expression have to be given initial values for the minimization procedure to begin. Different starting sets of values are likely to lead to different solutions with varying degrees of quality, and some times they may not lead to convergence.

We have applied this approach to the case where the allowed transformation is rigid (i.e. the hashing function is the one implied by Eqn. 1), and the points comprising the planar sets are assumed to be uniformly distributed over the unit disc. To the best of our knowledge, an analytical derivation of the expression for f_{UV} has not been possible to this date. However, given a discrete numerical approximation for this distribution, one can apply non-linear parametric fitting. In particular, using the method of Levenberg-Marquardt it can be determined that a good approximation is given by

$$f^*_{UV}(u,v) = \frac{1}{\left(\left(4.7u^2 + 3.9v^2\right)^2 + 36.7\right)^2}. \tag{6}$$

Approximating the probability density function instead of analytically deriving it is inconsequential to the methodology we are suggesting: indeed, in the following analysis f_{UV} would merely need to be replaced by f^*_{UV} (cf. Eqn. 6).

4 Framework of the Suggested Solution

In this section we discuss the framework that we are suggesting for redesigning the hashing function in order to guarantee a uniform distribution of the invariants that the function will produce.

In order to proceed with the method, knowledge of the joint probability density function f_{UV} or of an approximation f^*_{UV} is necessary. Determination of the expression for the joint pdf can be achieved either by fitting a parametric model to available real or synthetic data, or by calculating analytically the expected distribution based upon a model for the distribution of the model points. In order to simplify the notation, and without loss of generality, we will use f_{UV} to denote both f_{UV} and f^*_{UV}. And, of course, we will continue developing our approach in the context of the geometric hashing and of the hashing function of Eqn. 1.

Once the probability density function $f(u,v)$ is available, a transformation that maps the original distribution to the uniform distribution over a closed region (in particular, a rectangle) must be computed. This new mapping is effectively a *rehashing* function:

$$g: \Re^2 \rightarrow \Re^2,$$

and is used to evenly distribute the 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 sets of three points (cf. Eqn. 1) and generates invariant tuples (u',v') such that the equally-spaced hash buckets in this re-mapped space will have a uniform expected population.

Let us assume that we have a set of 2 one-dimensional, continuous, stochastic variables U and V. In the context of our problem, 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:

$$New\_u\,(U,V) : \Re^2 \longrightarrow \Re$$

and

$$New\_v\,(U,V) : \Re^2 \longrightarrow \Re\,.$$

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

$$f_{New\_U,New\_V}\,(New\_u, New\_v) = \int_{\Re^2} f_{UV}\,(U\,(New\_u, New\_v)\,, V\,(New\_u, New\_v))\,|J|^{-1}\,dUdV \tag{7}$$

where J is the Jacobian of the transformation $\mathbf{T}$; given the above assumptions the Jacobian is defined all over $\Re^2$ and is $\neq 0$. So the task at hand 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 but are fairly straightforward to compute. Occasionally, more than one mappings may produce the desired property. As a rule of thumb, one mapping that frequently can be computed easily and has the desired properties is the following:

$$\mathbf{T}(u_0, v_0) = \left(\int_{-\infty}^{u_0} \int_{-\infty}^{\infty} f(u,v)\ du\ dv\ ,\ \int_{-\infty}^{\infty} \int_{-\infty}^{v_0} f(u,v)\ du\ dv \right). \tag{8}$$

But as already stated other mappings will occasionally be possible. We will demonstrate this statement in the context of the hashing function (Eqn. 1) that we have been discussing. In essence, we will derive the mapping suggested above but after we have re-mapped the Cartesian coordinates to polar ones. The coordinates in the Cartesian domain are nothing but the invariants u and v which 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 Eqn. 4 in polar coordinates, we have that

$$f(\rho,\theta) = \frac{1}{3\pi\sigma^2} \cdot e^{-\left(\frac{\rho^2}{3\sigma^2}\right)}. \tag{9}$$

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 re-mapping $\mathbf{T}$. Notice that

$$\int_0^{\rho_0} \int_0^{2\pi} f(\rho,\theta)\ d\theta\ \rho d\rho = \left(1 - e^{-\frac{\rho_0^2}{3\sigma^2}}\right), \tag{10}$$

and that

$$\int_0^{\infty} \int_0^{\theta_0} f(\rho,\theta)\ d\theta\ \rho d\rho = \theta_0 \tag{11}$$

we can see that the re-mapping $\mathbf{T}$ in this case will be equal to

$$\mathbf{T}(u,v) = \left(1 - e^{-\frac{u^2+v^2}{3\sigma^2}}, \frac{(\operatorname{atan2}(v,u) + \pi)}{2\pi}\right) \tag{12}$$

where: (a) we have used the function atan2 instead of arctan because atan2 returns the phase in the interval $[-\pi,\pi]$, and (b) we have divided the atan2 component by 2π so that both components assume values between 0 and 1, and the re-mapped invariants will be uniformly distributed over the region $[0,1] \times [0,1]$ of the space of invariants. In the section where we describe the experiments, we will demonstrate the quality of the re-mapping suggested here by showing results on synthetic and real-world data. We conclude this section by noting that it suffices now 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 desired properties.

5 Additional Functions And Feature Distributions

In this section, we present additional results that were obtained by using the method outlined above, and for a variety of hashing functions and feature distribution combinations. So far, our discussion has assumed that the models to be recognized could only undergo rigid transformations on the plane, i.e. rotations and translations. Frequently, it is desirable to be able to recognize the models after they have also undergone a scaling transformation in addition to the rotation and translation (similarity transformation). The equation from which the hashing function is derived resembles Eqn. 1 and in fact is

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

In this expression, we used the notation associated with Figure 1. The last equation essentially 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 show that solving this equation for u and v provides two numbers that remain invariant under similarity transformations of the model.

The models may be allowed to undergo an affine transformation, the most general transformation on the plane which also incorporates "shear" in addition to rotation, translation and scaling. As expected, the relevant expression is different from the ones we have already presented. The most distinguishing difference has to do with the fact that 3 (not 2) points are needed now to form a basis:

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

Notice also that it is the distance from $\mathbf{q}$ to the barycenter of the triangle formed by the three points participating in the formation of the basis which is expressed as a linear combination of two distinct vectors.

As we have already mentioned, in addition to knowing the hashing function that is used to build the hash table, an assumption has to also be made regarding the distribution of the features that the hashing function will operate on. Below, we present results for several combinations of hashing function and feature distributions. Several of the results have been derived and presented previously and are only included to underline the generality of the method [27, 30]. On the other hand, those obtained using an assumption of Zipf-distributed feature points are new and will be discussed in more detail.

Table 5 shows several combinations of transformations and model feature distributions. For each of the shown combinations, we list the expression for the joint pdf of the invariants u and

Transform / Distribution	PDF of invariants	Rehashing Function Function
1 – Rigid / Gaussian	$f(u,v) = \frac{1}{3\pi\,\sigma^2} \cdot e^{-\left(\frac{u^2+v^2}{3\sigma^2}\right)}$	$h_1(u,v) = 1 - e^{-\frac{u^2+v^2}{3\sigma^2}}$ $h_2(u,v) = (\mathrm{atan2}(v,u)+\pi)/(2\pi)$
2 – Rigid / UnitDisc	$f(u,v) = \frac{1}{\left(\left(4.7u^2+3.9v^2\right)^2+36.7\right)^2}$	$h_1(u,v) = \frac{2}{\pi}\mathrm{atan}\left(\frac{(au)^2+\left(bv\right)^2}{c^2}\right) + \frac{2\left[(au)^2+\left(bv\right)^2\right]c^2}{\pi\left(\left[(au)^2+\left(bv\right)^2\right]^2+c^4\right)}$ $h_2(u,v) = (\mathrm{atan2}(v,u)+\pi)/(2\pi)$
3 – Similarity / Gaussian	$f(u,v) = \frac{12}{\pi\left(4\left(u^2+v^2\right)+3\right)^2}$	$h_1(u,v) = 1 - \frac{3}{4\left(u^2+v^2\right)+3}$ $h_2(u,v) = (\mathrm{atan2}(v,u)+\pi)/(2\pi)$
4 – Affine / Gaussian	$f(u,v) = \frac{2\sqrt{2}}{\pi\left(4u^2+4v^2+4uv+8/3\right)^{\frac{3}{2}}}$	$h_1(u,v) = 1 - \frac{2\sqrt{2}}{\sqrt{3}\sqrt{(4u^2+4uv+4v^2+8/3)}}$ $h_2(u,v) = \left(\mathrm{atan2}\left(u\sqrt{3}+v\sqrt{3}, u-v\right)+\pi\right)/(2\pi)$

Table 1: Various combinations of transformations and feature distributions. Note that $a = 4.7$, $b = 3.9$, and $c = 36.7^{1/4}$ in the second rehashing function.

v together with the re-mapping $\mathbf{T} = (h_1(u,v), h_2(u,v))$ that will rehash (u,v) to (u',v') so that the latter are uniformly distributed over a closed region of the space of invariants. It should be noted that the expression for the joint pdf in the case of rigid transformations and features that are uniformly distributed over the unit disc could not be derived analytically but rather through a numerical approximation. Details on the derivation of some of these expressions can be found in [27].

Data distributions frequently encountered in database applications follow the Zipf law [16]. In order to validate our methodology, we generated synthetic data whose features observed a two-dimensional Zipf distribution with constant k; the models 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 the steps described above. Derivation of an expression for the joint pdf by evaluation of the integral in Eqn. 3 proved impossible. Consequently, a numerical approach was attempted. We discovered that a good functional form to approximate the three distinct joint pdf's was given by

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

where the values of A, B, and C depended on the value of k used. In fact, through non-linear parameter estimation 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 easy to verify that the appropriate re-mapping equations are

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

This re-mapping 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 re-mapping that one could use. This second re-mapping 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:

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

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

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

As before, this re-mapping can be used in all three cases, after the appropriate values have been substituted for B, C and D. Table 5 shows the values for A, B, C and D as these have been determined by non-linear parameter estimation. In the experimental results section, we will show graphically the results and quality of this re-mapping for all three chosen values of 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

Table 2: The values of the parameters in the approximating expression, for various choices of the Zipf constant k.

6 Experimental Results

In the first subsection, we show results that demonstrate how even a very simple hashing function can lead to a non-uniform distribution of generated invariants. This is followed by a demonstration of how the quality of a hashing function can be misjudged if the hash table does not contain a sufficiently large number of entries. Finally, in the third subsection we discuss our experimental results.

6.1 Even Simple Hashing Functions Can Cause Problems

Let us assume the very simple case of a very large data set D that contains one-dimensional line segments that we are supposed to recognize independent of any translation transformations. The vertices of each line segment are selected uniformly from the unit interval $[-0.5, 0, 5]$

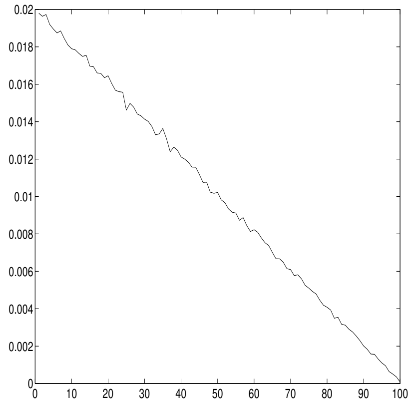


Figure 4: The probability density function for the length of the line segments; it is shown discretized over a set of 100 points.

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 such as the line segment's identity. The 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 several values from the range of possible values of h over others. This is demonstrated in Figure 4 which shows the discretized probability density function for h.

We next continue with a little more complicated example where the data set D contains two-dimensional triangular shapes which we wish to recognize independent of any rotation and/or translation and/or scaling transformation such a shape may have undergone; in other words, if Δ denotes one of the triangles in D, 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 D are drawn with uniform probability from the unit square $[-0.5, 0.5] \times [-0.5, 0.5]$.

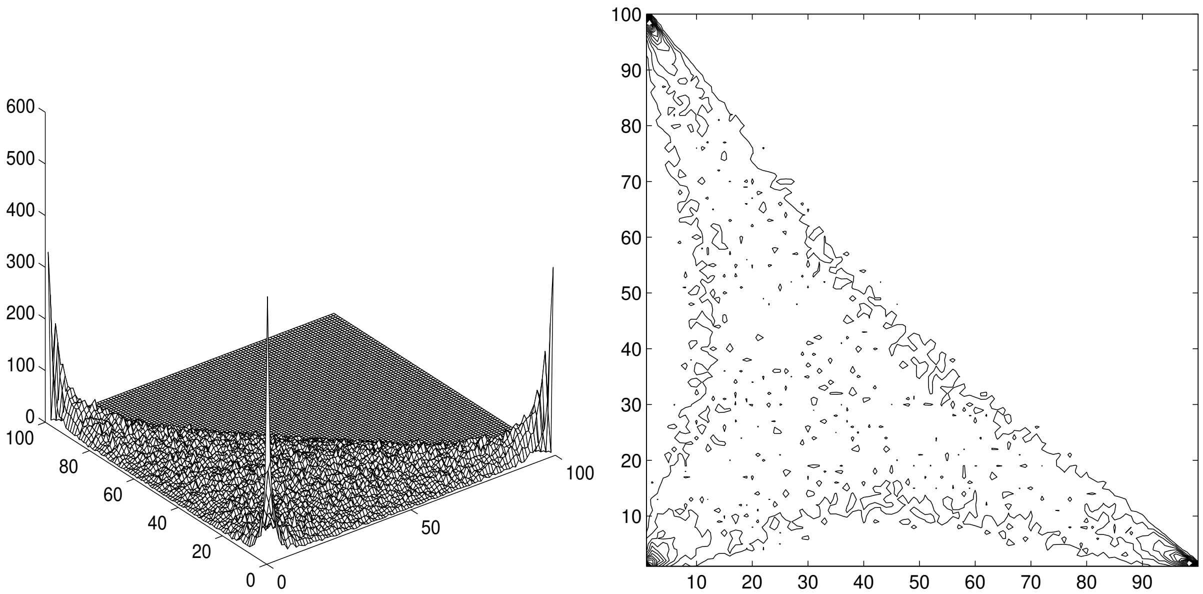


Figure 5: Mesh and contours for the probability density function for the angles of a triangle; the vertices of the formed triangles were uniformly distributed within the unit square.

Let us denote by $\left(x_{i_j}, y_{i_j}\right)$, (with $j = 1, 2, 3$) the three vertices of the i-th model triangle that is in the data set D. 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 D, expressed in radians, and q denotes that the values of the 2-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 would be 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. Figure 5 shows the mesh and contours of the discretized joint probability density function for this second hashing function.

6.2 Misjudging The Quality Of A Hashing Function

For the purposes of this demonstration, we have generated the meshes for the occupancies of the hash table buckets for four different values of stored entries. The used hashing function was kept the same during the experiments but the number of hash entries increased by an order of magnitude each time. The intention here is to show that it is possible at times to misjudge the quality of an employed hash function.

Figures 6, 7, 8, 9 show the meshes for 1,200, 12,000, 120,000, and 1,200,000 entries stored in the database respectively. The scaling of the axes is the same in all four cases in order to show the progressive appearance of statistical behavior as the number of entries increases. Allowed transformation are rigid and the points were assumed to be Gaussianly distributed.

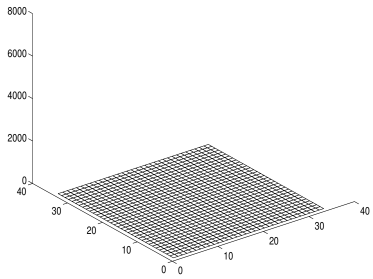


Figure 6: Mesh for 1,200 entries.

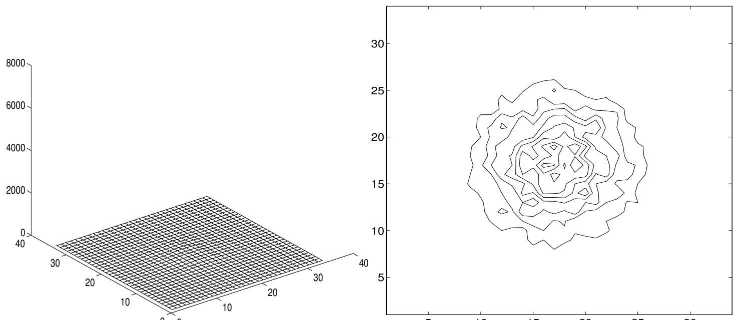


Figure 7: Mesh and contours for 12,000 entries.

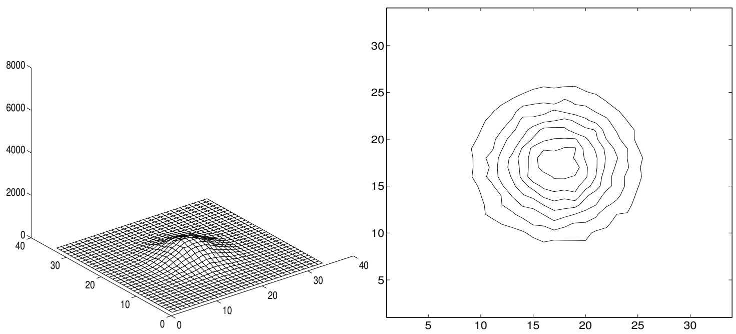


Figure 8: Mesh and contours for 120,000 entries.

Of course, and for hash functions of different dimensionality, the number of stored entries required before this statistical behavior reveals itself is expected to also be different.

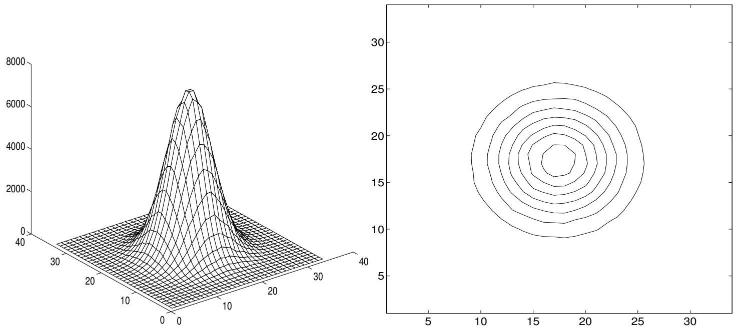


Figure 9: Mesh and contours for $1,200,000$ entries.

6.3 The Efficiency Of The Suggested Framework

Our discussion so far has proceeded independent of data objects that would be encountered in the context of a database. Instead, 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 will now present results from experiments that used both a synthetically–generated and a real-world database.

In particular, we have carried out our experiments using a set of synthetically-derived two-dimensional polygonal shapes. We have concentrated on the vertices of these shapes; the coordinates of these vertices were drawn from a two-dimensional Gaussian distribution $N\left(0, \begin{pmatrix} \sigma & 0 \\ 0 & \sigma \end{pmatrix}\right)$. We used the vertex coordinates in the manner prescribed by the geometric hashing approach and made multiple 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 two different hashing functions (i.e. 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 results have been obtained through a Monte Carlo simulation with synthetic data. The height of the mesh at a given 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 derived after we apply the h functions always contain 97.5% of the total number of entries that the models generated. Naturally, the figures corresponding to the *synthesis*, $g \circ h$, of functions contain all of the entries generated by the models.

Figure 10 shows the meshes for the occupancies of the buckets for the hash tables corresponding to the first case of Table 5. For clarity we will be referring to the function $g \circ h$ of the i-th case of that Table by $(g \circ h)_i$.

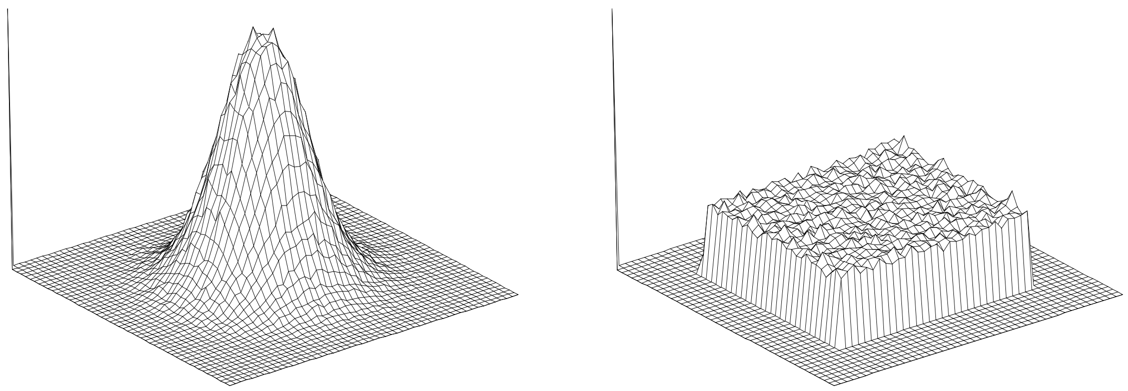


Figure 10: (a) Mesh for the *pdf* corresponding to the hashing function of case 1 (rigid transformation). (b) Mesh for the *pdf* corresponding to $(g \circ h)_1$. The features of the various models followed a Gaussian distribution.

Figure 11 shows the meshes for the occupancies of the buckets for the hash tables corresponding to the second case of Table 5. Note that here, there was no analytical derivation of the joint pdf but rather a numerical approximation was carried out; as can be verified from the two meshes, the improvement obtained by the suggested methodology is dramatic.

Figures 12 and 13 show the meshes for the occupancies of the buckets for the hash tables corresponding to the remaining two cases (3 and 4) of Table 5. Once again, the usefulness of our methodology is clearly evident.

Next, we present the results we obtained from our Monte-Carlo simulations on large databases of objects where the features were distributed following a Zipf distribution. In particular, the x and y coordinates of the feature points were drawn independently from three different Zipf distributions which corresponded to k values of 0.1, 0.2 and 0.5. The formulas from the previous section were used to produce the hash tables for h and $g \circ h$. The models were allowed to undergo similarity

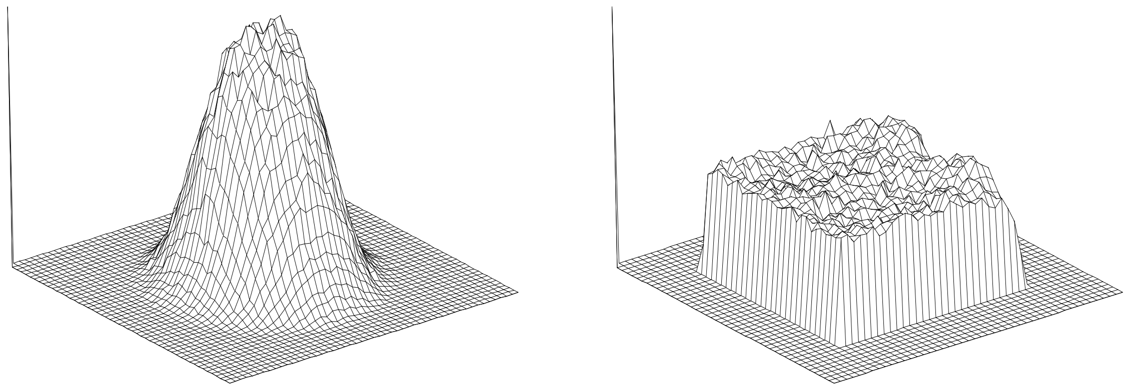


Figure 11: (a) Mesh for the *pdf* corresponding to the hashing function of case 2 (rigid transformation). (b) Mesh for the *pdf* corresponding to $(g \circ h)_2$. The features of the various models were distributed uniformly over the unit disc.

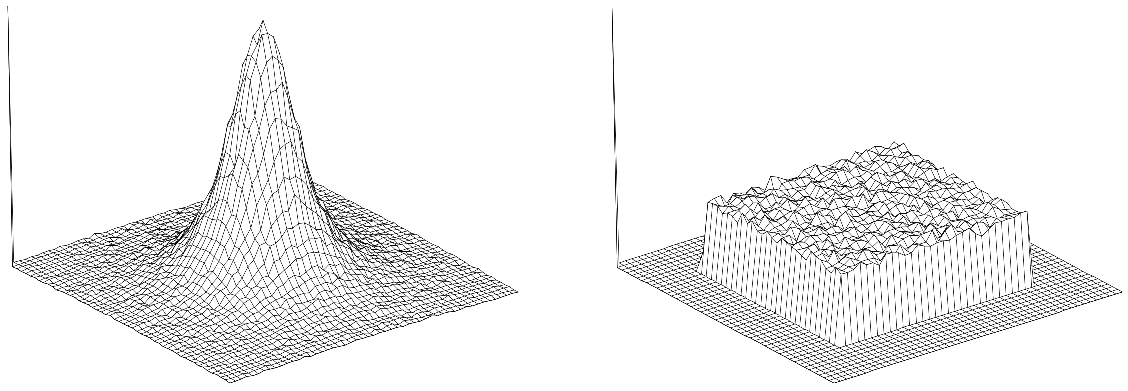


Figure 12: (a) Mesh for the *pdf* corresponding to the hashing function of case 3 (similarity transformation). (b) Mesh for the *pdf* corresponding to $(g \circ h)_3$. The features of the various models followed a Gaussian distribution.

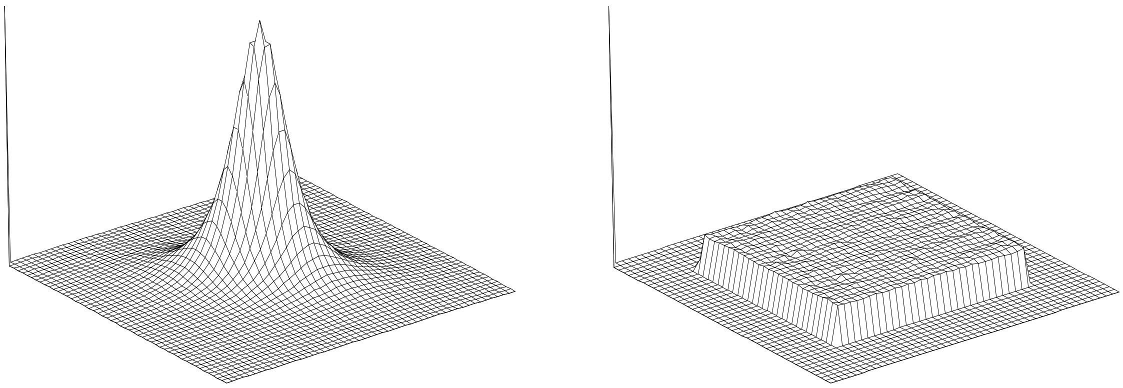


Figure 13: (a) Mesh for the *pdf* corresponding to the hashing function of case 4 (affine transformation). (b) Mesh for the *pdf* corresponding to $(g \circ h)_4$. The features of the various models followed a Gaussian distribution.

transformations.

Figure 14 shows the meshes for the occupancies of the buckets for the hash tables corresponding to the hashing function of Eqn. 13 (similarity transformation) and to $g \circ h$ in the case where $k = 0.1$. Figures 15 and 16 show similar results for the other two cases: $k = 0.2$ and $k = 0.5$.

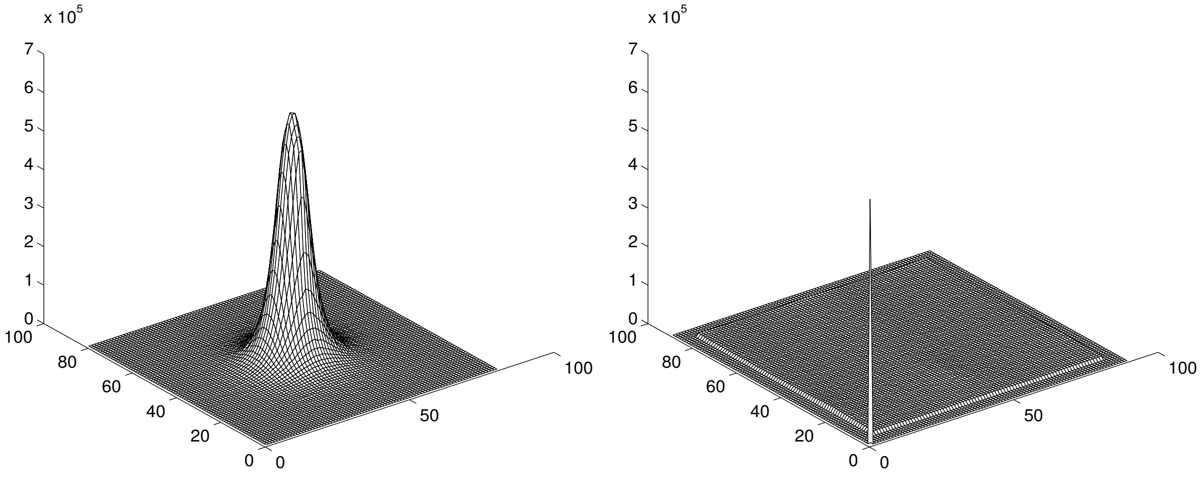


Figure 14: (a) Mesh for the *pdf* corresponding to the similarity transformation, with the features following a Zipf distribution with $k = 0.1$ (b) Mesh for the *pdf* mesh corresponding to $g \circ h$.

Finally, we show results from using a real database. In particular, the models to be recognized were fingerprints of 600 individuals. Each fingerprint was represented as a collection of *minutiae*, i.e.

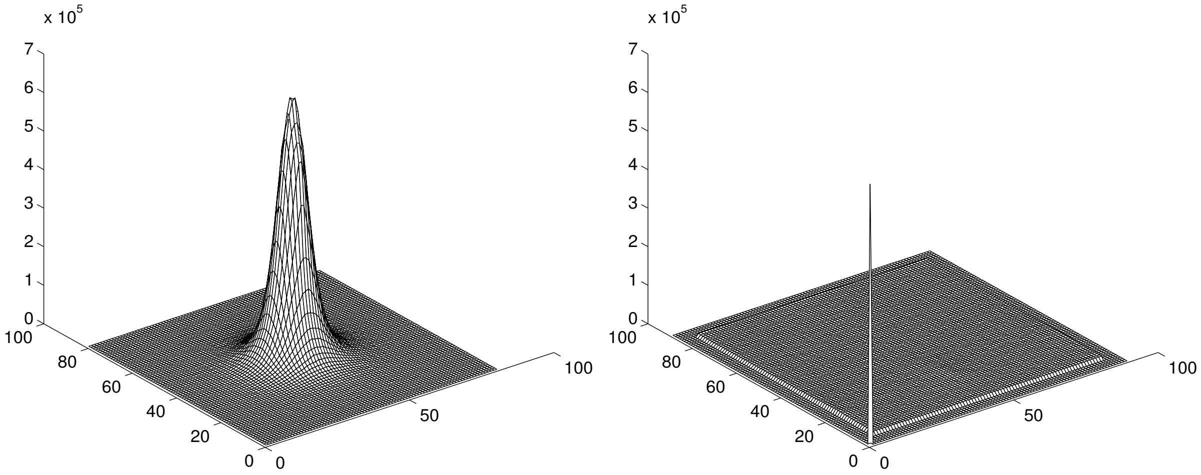


Figure 15: (a) Mesh for the *pdf* corresponding to the similarity transformation, with the features following a Zipf distribution with $k = 0.2$ (b) Mesh for the *pdf* mesh corresponding to $g \circ h$.

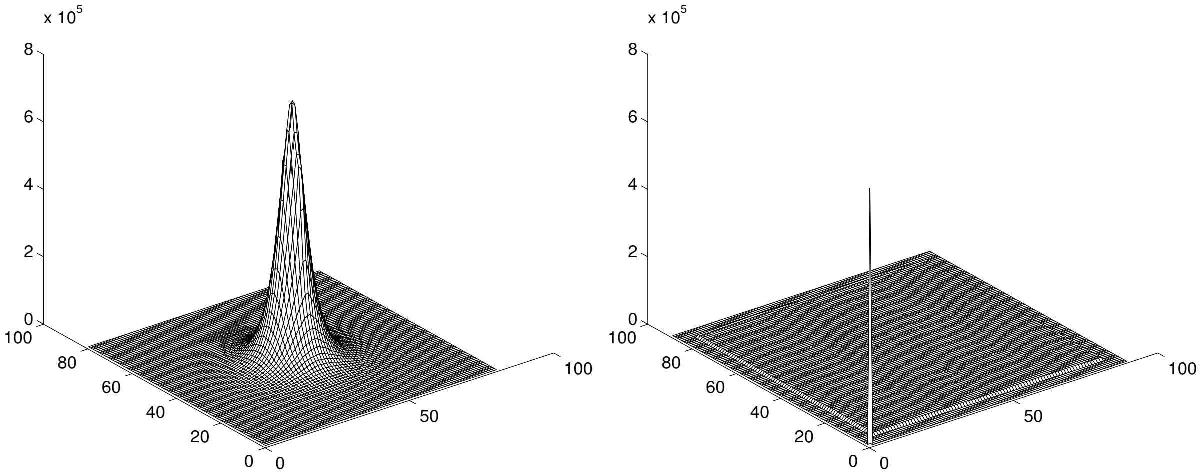


Figure 16: (a) Mesh for the *pdf* corresponding to the similarity transformation, with the features following a Zipf distribution with $k = 0.5$ (b) Mesh for the *pdf* mesh corresponding to $g \circ h$.

points of interest corresponding to the locations of things like bifurcations, swirls, ridge terminations etc. The only information available was the positional information about a fingerprint's minutiae. In that respect, the models were not any different than the model M used in the development of our discussion. The allowed transformation was 2D-affine and the feature points were distributed uniformly over the image square (512 pixels x 512 pixels).

What makes this case a little 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 Eqn. 14 exhibits some peculiar properties whose study escapes the scope of this discussion. In related work [28], we study an alternative affine hashing function and how it behaves when the feature domain is convex and unspecified. We have successfully applied the methodology that was developed in the present discussion and are showing below the meshes for this alternative hashing function as well as the corresponding $g \circ h$. Due to the fact that both the description of the alternative hashing function as well as of the remapping $g \circ h$ are fairly involved and lengthy, the reader is referred to [28] for more details. Figure 17 shows the meshes for the hashing function used and the corresponding $g \circ h$. Notice that unlike the hashing functions that we have examined so far, this alternative affine-invariant hashing function exhibits a discontinuity. However, our methodology applies in a straightforward manner and the results are rewarding.

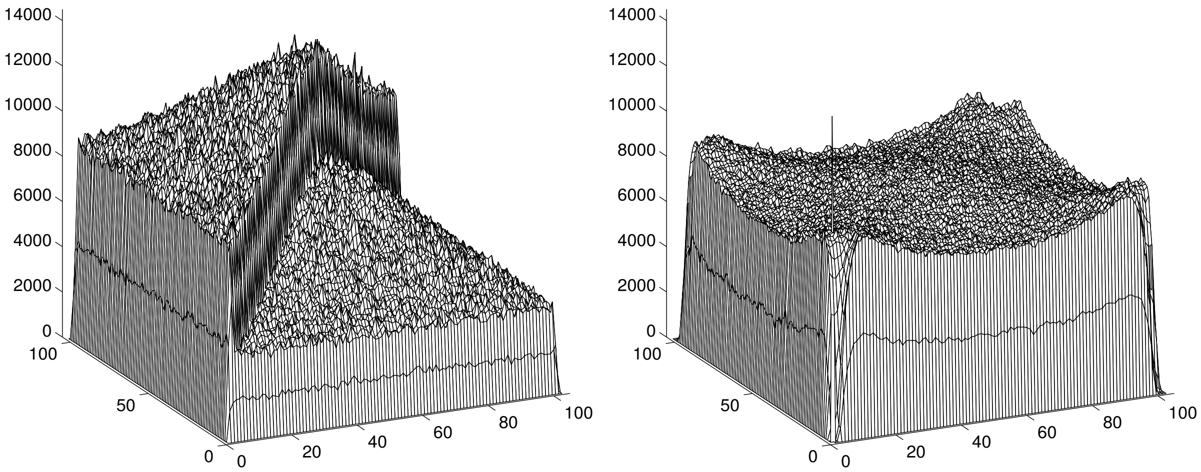


Figure 17: (a) Mesh for the *pdf* corresponding to the affine-invariant hashing function mentioned in the text. The database used contains fingerprints from 600 individuals and the features under consideration are the fingerprint *minutiae* together with their positional information. The features were uniformly distributed over the image square (512 pixels $\times$ 512 pixels). (b) Mesh for the *pdf* mesh corresponding to $g \circ h$.

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 are discussed in [19]. In [26], a number of hashing functions are tested and their performance verified against data of very modest sizes. Bentley and Friedman [2] provide a definition of the range-search problem on multiple attributes which extends to more than one dimensions. Multi–dimensional data (known also as k–d data) are accessed and processed by representations in lower dimensions. A large number of approaches take the view that k–d data can be transformed in a step–wise manner to one dimension representations and thus traditional access methods can be used for their manipulation. Gutman's R–trees and variations [14, 34] use this idea. The hB-tree [31] 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 acquire a severe skew, thus affecting search time. In [9], 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 [17]. The usual assumptions of uniformity and independence are contended in [10]. The concept of the fractal dimension is proposed as the means to describe the deviation from uniformity. Guenther and Buchmann [13] 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 [7]. Bastani et al. [8] address the issue of reliability in distributed indexing. In [18], a structure called the "Multiplexed R-tree" is proposed; this is essentially a single R-tree with pointers spanning disks. Honishi et. al. [15] proposed a method where the index structure is partitioned into subtrees indexed by a hash structure. Matsliach and Shmueli [24] presented a parallel access method suitable for shared-memory multiprocessor systems. Krishna et al. [33] propose the use of the dB-tree, which is based on the B-link tree [22], 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 [20] simplify the distribution of nodes among processors.

In many instances deterioration of the performance happens due to uneven load distribution [5, 32]. It has been also shown in [23] that the probability to find idle computing elements in a group of nodes is very high. In the context of our work, our objective has been to have the various database entries distribute evenly among the 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 set of hash table buckets among them. Desirable scalability properties follow immediately since the probability density function f_{goh} of the invariants is constant over the entire *finite* region of the new produced values. This immediately 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 non-uniform bucket occupancy distributions in the hash tables. In this paper, we presented a two-stage methodology used to re-map data sets so that the resulting hashing scheme exhibit competitive performance characteristics. We begin with the knowledge of 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 well-defined methods from probability theory to *re-design* the hashing function so that it exhibits a set of desired properties.

While there is a wealth of hashing techniques and distributed/parallel access methods, none of these techniques deals with the probabilistic behavior of hashing indices. By introducing our framework based on geometric hashing 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 objects based on multiple key "features" in hash-table access structures. The large number of objects used exposes the statistical behavior of the bucket occupancy patterns.
- Use of geometric hashing to create the invariant descriptors of the objects (as they are rotated, translated, and scaled on the plane) and exploitation of the resulting occupancy distributions in order to generate hash access tables with close-to-uniform occupancy patterns.
- Easy application of our framework in a parallel and multi-computer architectures to solve realistic problems offering very short response times.

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

The resulting system exhibits the desired scalability properties. 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.

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

References

[1] D. Ballard and C. Brown. *Computer Vision.* Prentice Hall, Englewood Cliffs, NJ, 1982.

[2] J. L. Bentley and J. H. Friedman. Data Structures for Range Searching. *ACM Computing Surveys*, 11(4):397–409, 1979.

[3] A. Califano and R. Mohan. Multidimensional Indexing for Recognizing Visual Shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16(4):373–392, 1994.

[4] A. Califano and I. Rigoutsos. FLASH: A Fast Look-Up Algorithm for String Homology. *Computational Approaches for Biological Systems Journal*, To appear.

[5] T. Casavant and J. Kuhl. A Taxonomy of Scheduling in General–Purpose Distributed Computing Systems. *IEEE Transactions on Software Engineering*, 14(2), February 1988.

[6] M. Costa, R. Haralick, and L. Shapiro. Optimal Affine-Invariant Matching: Performance Characterization. In *Proceedings SPIE Conference on Image Storage and Retrieval*, San Jose, CA, January 1992.

[7] C. Ellis. Distributed Data structures: A Case Study. *IEEE Transactions on Computers*, C-34(12):1178–1185, 1985.

[8] S. Iyengar F. Bastani and I. Yen. Concurrent maintenance of data structures in a distributed environment. *The Computer Journal*, 31(12):165–174, 1988.

[9] C. Faloutsos. Gray Codes for Partial Match and Range Queries. *IEEE Transaction on Software Engineering*, 14(10), October 1987.

[10] C. Faloutsos and I. Kamel. Beyond Uniformity and Independence: Analysis of R–trees Using the Concept of Fractal Dimension. In *Proceedings of the 1994 ACM PODS Conference on Principles of Database Systems*, Minneapolis, Minnesota, 1994.

[11] P. Flynn and A. Jain. $3D$ Object Recognition Using Invariant Feature Indexing of Interpretation Tables. *Computer Vision, Graphics, and Image Processing: Image Under standing*, 55(2):119–129, 1992.

[12] D. Gavrila and F. Groen. $3D$ Object Recognition from $2D$ Images Using Geometric Hashing. *Pattern Recognition Letters*, 13(4):263–278, 1992.

[13] O. Guenther and A. Buchmann. Research Issues in Spatial Databases. *ACM SIGMOD Record*, 19(4):61–67, 1990.

[14] A. Guttman. R-Trees: A Dynamic Index Structure for Spatial Searching. In *Proceedings of 1984 SIGMOD conference*, pages 47–57, 1984.

[15] T. Honishi, T. Satoh, and U. Inoue. An Index Structure for Parallel Database Processing. In *IEEE Second International Workshop on Research Issues on Data Engineering*, pages 224–225, 1992.

[16] Y. Ioannidis. Universality of Serial Histograms. In *Proceedings of the 19th VLDB Conference*, Dublin, Ireland, 1993.

[17] H.V. Jagadish. Linear Clustering of Objects with Multiple Attributes. In *Proceedings of the 1990 ACM SIGMOD Conference on the Management of Data*, Atlantic City, NJ, 1990.

[18] I. Kamel and C. Faloutsos. Parallel R-Trees. In *Proceedings of ACM SIGMOD Conference*, pages 195–204, 1992.

[19] G. Knott. Hashing Functions. *The Computer Journal*, 18(3), March 1975.

[20] B. Kroll and P. Widmayer. Distributing a Search Structure Among a Growing Number of Processors. In *Proceedings of ACM SIGMOD Conference*, pages 265–276, 1994.

[21] Y. Lamdan. *Geometric Hashing.* PhD thesis, New York University, June 1989.

[22] P. Lehman and S. Yao. Efficient locking for concurrent operations on b-trees. *ACM Transactions on Database Systems*, 6(4):650–670, 1981.

[23] M. Livny and M. Melman. Load Balancing in Homogeneous Broadcast Distributed Systems. *ACM Performance Evaluation review*, 11(1), Spring 1982.

[24] G. Matsliach and O. Shmueli. An efficient method for distributing search structures. In *First International Conference on Parallel and Distributed Information Systems*, pages 159–166, 1991.

[25] R. Mohan, D. Weinshall, and R. Sarukkai. 3*D* Object Recognition By Indexing Structural Invariants From Multiple Views. In *Proceedings of the International Conference on Computer Vision*, pages 264–268, Berlin, Germany, May 1993.

[26] M.V. Ramakrishna. Hasing in practice, Analysis of Hashing and Universal Hashing. In *Proceedings of the 1988 ACM-SIGMOD Conference on the Management of Data*, Chicago, IL, 1988.

[27] I. Rigoutsos. *Massively Parallel Bayesian Object Recognition.* PhD thesis, New York University, August 1992.

[28] I. Rigoutsos. A New Scheme For Generating Uniformly-Distributed 2D Affine Invariants And WHich Can Be Fine Tuned To Any Convex 2D Feature Domain. Submitted for publication, 1996.

[29] I. Rigoutsos and A. Califano. Searching In Parallel For Similar Strings. *IEEE Computational Science and Engineering*, pages 60–75, Summer 1994.

[30] I. Rigoutsos and A. Delis. The Impact of Hashing Function Choice On The Load Balancing in Very Large Datasets. In *Proceedings of the 8th International SEKE Conf.*, Lake Tahoe, Nevada, 1996.

[31] B. Salzberg. Practical Spatial Database Access Methods. In *IEEE Symposium on Applied Computing*, pages 82–90, 1991.

[32] A. Svensson. History, and Intelligent Load Sharing Filter. In *Proceedings of the 10th International Conference on Distributed Computing Sustems*, 1991.

[33] P. Krishna T. Johnson and A. Colbrook. Distributed Indices for Accessing Distributed Data. In *12th IEEE symposium on Mass Storage Systems*, pages 199–207, 1993.

[34] N. Roussopoulos T. Sellis and C. Faloutsos. The R+-tree: A Dynamic Index for Multi-dimensional Objects. In *Proceedings of the 13th VLDB Conference*, pages 507–518, 1987.

[35] M. Tsangaris and J. Naughton. A Stochastic Approach for Clustering in Object Bases. In *Proceedings of the 1991 ACM-SIGMOD Conference on the Management of Data*, Denver, CO, 1991.

[36] J.T.L. Wang, G.W. Chirn, T.G. Marr, B. Shapiro, D. Shasha, and K. Zhang. Combinatorial Pattern Discovery for Scientific Data: Some Preliminary Results. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, Minneapolis, MN, May 1994.

[37] J.T.L. Wang, K. Zhang, K. Jeong, and D. Shasha. A System for Approximate Tree Matching. *IEEE Transactions on Knowledge and Data Engineering*, 6(4), August 1994.