# Metric Databases

Edgar Chávez
Escuela de Ciencias Físico Matemáticas
Universidad Michoacana, Mexico

Gonzalo Navarro
Dept. of Computer Science
University of Chile

The advent of the digital age creates an increasing interest to search for information in large unstructured repositories containing textual and multimedia data. Retrieval from those repositories requires more powerful query languages, which exceed the querying capabilities of traditional database technologies. The metric space model, described herein, is an extension of the exact searching paradigm aiming to cope with the new requirements.

Traditional databases are built around the concept of exact searching: the database is divided into *records*, each record having a fully comparable *key*. Queries to the database return all the records whose keys match the search key *exactly*. More sophisticated searches such as range queries on the numerical keys or prefix searching on alphabetical keys still rely on the concept of exact comparison. Recently, Database Manager Systems (DBMS) incorporate the ability of storing so called "multimedia objects", which nevertheless can rarely be searched by content.

The most distinguishing feature of multimedia objects is that there is no point in comparing them by exact equality. Rather, users are interested in how similar two objects are. The metric space model gives a theoretical foundation to define a meaningful way to retrieve multimedia data. A *metric database* is a collection of digital objects (of any kind) with a *perceived* similarity and a formal way to compute this perceived similarity as a metric. The perceived similarity is provided by everyday experience or by experts.

The theoretical foundations for Metric Databases are well established. However, metric databases are not yet in a mature state. The end users do not have the equivalent of a full DBMS for multimedia indexing. There is, however, at least one rather robust, open source implementation of an index for metric spaces [6].

## 1 Applications of Metric Databases

Although retrieving multimedia data is one of the main applications for metric databases, these extend well beyond. We describe now several of those [5].

**Query by Content in Multimedia Objects.** New data types such as images, fingerprints, audio and video (called "multimedia" data types) cannot be meaningfully queried in the classical sense. Not only they cannot be ordered, but they cannot even be compared by equality. No application will be interested in searching an audio segment exactly equal to a given one. The probability that two different images are pixel-wise equal is negligible unless they are digital copies of the same source. In multimedia applications, all the queries ask for objects similar to a given one. Some example applications are face recognition, fingerprint matching, voice recognition, image retrieval, etc.

These approaches are based on the definition of a similarity function among objects. Those functions are provided by an expert, but they pose no assumptions on the type of queries that can be answered. In many cases, the distance is obtained via a set of $k$ "features" which are extracted from the object (e.g., in an image a useful feature is the color histogram). Then each object is represented as its $k$ features, that is, a point in a $k$-dimensional space.

**Information Retrieval.** Although not considered a multimedia data type, unstructured text retrieval poses similar problems as multimedia retrieval. This is because textual documents are in general not structured to easily provide the desired information. Text documents may be searched for strings that are present or not, but in many cases they are searched for semantic concepts.

The problem is basically solved by retrieving documents similar to a given query. The user can even present a document as a query, so that the system finds similar documents. Some similarity approaches are based on mapping a document to a vector of integer values, so that each dimension is a vocabulary word and the number of times the word appears in the document is the coordinate of the document in that dimension. Similarity functions are then defined in that space. Notice however that the dimensionality of the space is very high (thousands of dimensions).

**Text Retrieval.** Another problem related to text retrieval is spelling. Since huge text databases with low quality control are emerging (e.g., the Web), and typing, spelling or OCR (optical character recognition) errors are commonplace in the text and the query, we have that documents which contain a misspelled word are no longer retrievable by a correctly written query. Models of similarity among words exist (variants of the "edit distance") which capture those kind of errors. In this case, we give a word and want to retrieve the words close to it.

**Computational Biology.** ADN and protein sequences are the basic object of study in molecular biology. As they can be modeled as text, we have the problem of finding a given sequence of characters inside a longer sequence. However, an exact match is unlikely, and computational biologists are more interested in finding parts of a longer sequence which are similar to a given short sequence. The fact that the search is not exact is due to minor differences in the genetic streams that describe sequences of similar functionality, evolution, experimental errors, and so on. The measure of similarity used is related to the type of differences one is interested in.

**Pattern Recognition and Function Approximation.** A simplified definition of pattern recognition is the construction of a function approximator. One has a finite sample of the data, and each data sample is labeled as belonging to a certain class. When a fresh data sample is provided, one must label this new sample as belonging to one of the known classes.

If the objects are $m$-dimensional vectors of real numbers then a natural choice is neural nets and fuzzy function approximators. There is also another popular universal function approximator: the $k$-nearest neighbor classifier. It consists of finding the $k$ nearest objects to the unlabeled sample, and assigning to this object the label having majority among the $k$ nearest neighbors. Opposed to neural nets and fuzzy classifiers, the $k$-nearest neighbor rule has zero training time, but if no indexing algorithm is used it has linear complexity. Other applications of this universal function approximator are density estimation and reinforcement learning. In general, in any application where we want to infer a function based on a finite set of samples we have a potential application.

**Audio and Video Compression.** Audio and video transmission over a narrow-band channel is an important problem, for example in internet-based audio and video conferencing. A frame (a static picture in a video, or a fragment of the audio) can be thought of as formed by a number of (possibly overlapped) sub frames ($16 \times 16$ subimages in a video, for example). In a very succinct description, the problem can be solved by sending the first frame as is and, for the next frames, sending only the subframes having a "large difference" from the previously sent subframes. This description encompasses the MPEG standard.

This poses the need to efficiently finding subframes similar to the one that is to be sent, among a repository of recently sent subframes.

## 2 The Metric Space Model

Proximity queries are those extensions of the exact searching where we want to retrieve objects from a database that are *close* to a given query object [5]. The query object is not necessarily a database element. The concept can be formalized using the metric space model, where a distance function $d(x, y)$ is defined for every site in a set $\mathbb{X}$. The distance function $d$ has *metric* properties, that is, it satisfies $d(x, y) \geq 0$ (positiveness), $d(x, y) = d(y, x)$ (symmetry), $d(x, y) = 0$ iff $x = y$ (strict positiveness), and $d(x, y) \leq d(x, z) + d(z, y)$ (triangle inequality). The latter is the key property permitting solutions better than brute-force.

The database is a set $\mathbb{U} \subseteq \mathbb{X}$, and the query object, $q$, is an arbitrary element of $\mathbb{X}$. A similarity query can be of two basic types: A *Metric Range Query* $(q, r)_d = \{u \in \mathbb{U} \ : d(q, u) \leq r\}$, which retrieves all database elements at distance at most $r$ to $q$; and a *K Nearest Neighbor Query* $nn_k(q)_d = \{u_i \in \mathbb{U} \ : \ \forall v \in \mathbb{U}, \ d(q, u_i) \leq d(q, v) \ and \ |\{u_i\}| = k\}$, which retrieves the $k$ database elements closest to $q$.

Any such query can be answered by comparing the query $q$ against the whole database, that is, computing $d(q, u)$ for every $u \in \mathbb{U}$. As this is prohibitive, *indexing techniques* have been proposed to save as many distance computations as possible. Depending on the application, attention has also to be paid to I/O costs and other CPU costs beyond computing distances.

There are some proximity searching problems where a distance cannot be defined. Instead, a *(dis)similarity* is defined, which is a function not obeying the triangle inequality. It is common that the dissimilarity obeys a relaxed version of the triangle inequality, namely $d(x, y) \leq \nu(d(x, z) + d(z, y))$, with $\nu \geq 1$, with probability $\epsilon > 0$. Those cases can be dealt with variants of the techniques used for metric spaces. If we cannot have even the relaxed version of the triangle inequality, then no better-than-brute-force approaches can be used.

## 3 The Curse of Dimensionality

Vector spaces, using typically the Euclidean distance, can be seen as metric spaces where objects are arrays of numbers. There exist a lot of indexing mechanisms specifically aimed at indexing vector spaces [7]. In general they obtain better performance than general metric space indexes, especially when the space has few coordinates. However, the situation gets more complicated with higher dimensional spaces [1].

It is well known that the performance of any algorithm on a uniformly distributed vector space will suffer from an exponential dependency on the dimension of the space. A random query with nonempty outcome will force any index to scan the whole database, rendering it useless. This is called the "curse of dimensionality" and makes uniformly distributed spaces of dimension 20 or more, intractable in practice. On the other hand, if the data points are not uniformly distributed in the space, then some indexes are able to perform better by mapping the space to less dimensions without significantly losing distance information. The reason is that the *representational dimension* of the space (number of coordinates) can be much larger than the *intrinsic dimension*, which is, intuitively, the lowest dimension where the data points could be properly represented.

One of the benefits of seeing a vector space as a metric space is that the performance of the indexes directly depends on the intrinsic rather than the representational dimension, without need of any mapping. Even in metric spaces, where there is no notion of coordinates, there are spaces that are intrinsically more difficult to search than others, independently of the indexing method used. In order to accurately predict the achievable performance of searching a given metric space, and to apply the most adequate index given its intrinsic difficulty, it is important to be able of quantifying this intrinsic difficulty [5].

Let us start with a well-known example. Consider a distance such that $d(x,x) = 0$ and $d(x,y) = 1$ for all $x \neq y$. Under this distance (in fact an equality test), we do not obtain any information from a comparison except that the element considered is or is not our query. It is not possible to avoid a sequential search in this case, no matter how smart we made the algorithm.

Let us consider the *histogram* of distances between points in the metric space $\mathbb{X}$. This can be approximated by using the dictionary $\mathbb{U}$ as a random sample of $\mathbb{X}$. The idea is that, as the space has higher intrinsic dimension, the mean $\mu$ of the histogram grows and/or its variance $\sigma^2$ is reduced (at least this is the case on random vector spaces). Our previous example is an extreme case.

Figure 1 gives an intuitive explanation of why the search problem is harder when the histogram is concentrated. If we consider a random query $q$ and its comparison against some database element $p$, the distance $d(p,q)$ is distributed according to the histogram of the figure. The triangular inequality tells us that we can discard any point $u \in \mathbb{U}$ such that $d(p,u) \notin [d(p,q) - r, d(p,q) + r]$. The grayed areas in the figure show the points that we cannot discard. As the histogram is more and more concentrated around its mean, less and less points can be discarded using the information given by $d(p,q)$. Moreover, in many cases (e.g., to retrieve a fixed fraction of the database) the search radius $r$ must grow as the mean distance $\mu$ grows, which makes the problem even harder.

This phenomenon is independent of the nature of the metric space (vectorial or not, in particular) and gives us a way to quantify how hard is to search on an arbitrary metric space. A very rough but useful definition of intrinsic dimension is $\rho = \frac{\mu^2}{2\sigma^2}$. It grows as the median grows or the variance shrinks, as desired. Moreover, when applied to uniformly distributed vector spaces it matches the classic concept of representational dimension very well, within a constant factor between 1.0 and 1.5 that depends on the type of distance used. Finally, measuring this intrinsic dimension on an arbitrary and unknown metric space can be accomplished by simple statistical means via a reasonable number of distance evaluations among random points of the set. This is much simpler and cheaper than other approaches.

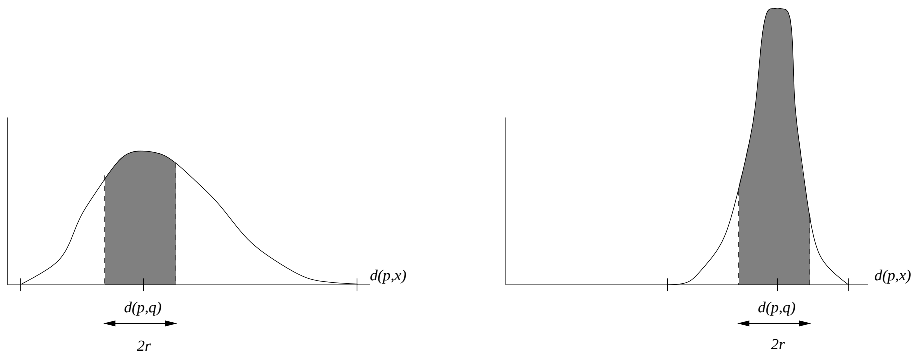

Figure 1: A low-dimensional (left) and high-dimensional (right) histogram of distances, showing that on high dimensions virtually all the elements become candidates for exhaustive evaluation.

On the other hand, the measure $\rho$ approximates only roughly the real difficulty of searching a metric space. Several other measures have been proposed with variable success and easiness of use, and the issue is not solved at all. Moreover, the most relevant open issue in metric database searching is how to cope with spaces of high dimension.

## 4 Current Algorithms for Metric Databases

Different indexing schemes have been proposed to speed up similarity searches [5]. The idea is to use the triangle inequality to filter out database elements $u$ that can be proved to be far enough from $q$ without the need to compute $d(q, u)$.

### 4.1 Pivoting Schemes

A *pivot* is a distinguished database element, whose distance to some other elements has been precomputed and stored in an index. Imagine that we have precomputed $d(p, u)$ for some pivot $p$ and every $u \in \mathbb{U}$. At search time, we compute $d(p, q)$. Then, by the triangle inequality, $d(q, u) \geq |d(p, q) - d(p, u)|$, so if $|d(p, q) - d(p, u)| > r$ we know that $d(q, u) > r$, hence $u$ will be filtered out without need of computing that distance. Figure 2 illustrates.

The most basic scheme chooses $k$ pivots $p_1 \ldots p_k$ and computes all the distances $d(p_i, u)$, $u \in \mathbb{U}$, into a table of $kn$ entries. Then, at query time, all the $k$ distances $d(p_i, q)$ are computed and every element $u$ such that $D(q, u) = \max_{i=1\ldots k} |d(p_i, q) - d(p_i, u)| > r$ is discarded. Finally, $q$ is compared against the elements not discarded.

As $k$ grows, we have to pay more comparisons against pivots, but $D(q, u)$ becomes closer to $d(q, u)$ and more elements may be discarded. It can be shown that there is an optimum number of pivots $k^*$, which grows fast with the dimension and becomes quickly unreachable because of memory limitations. In all but the easiest metric spaces, one simply uses as many pivots as memory permits. There exist many variations over the basic idea, including different ways to sort the table of $kn$ entries to reduce extra CPU time, e.g. [2].

Although they look completely different, several tree data structures are built on the same pivoting concept, e.g. [9]. In most of them, a pivot $p$ is chosen as the tree root, and its subtrees

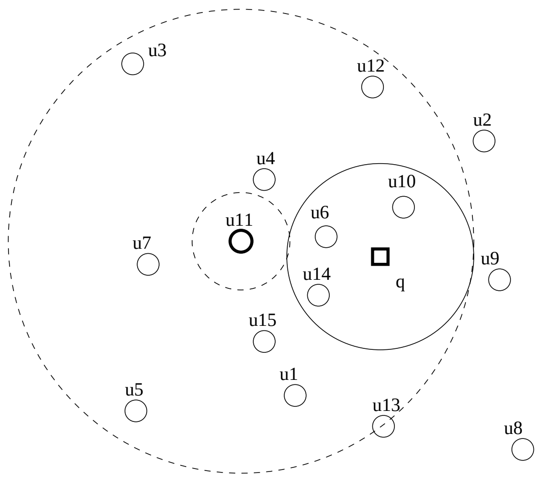

Figure 2: Using pivots to discard elements. Given query $q$ with the radius indicated by the solid circle, we should output $u_6$, $u_{10}$ and $u_{14}$. We measure $d(q, u_{11})$. Knowing the distances from $u_{11}$ to the other elements, we are able of discarding $u_2$, $u_8$ and $u_9$ without comparing them against $q$.

correspond to ranges of $d(p, u)$ values, being recursively built on the elements they have. In some cases the exact distances $d(p, u)$ are not stored, just the range can be inferred from the subtree the element $u$ is in. Albeit this reduces the accuracy of the index, the tree usually takes $O(n)$ space instead of the $O(kn)$ needed with $k$ pivots. Moreover, every internal node is a partial pivot (which knows distances to its subtree elements only), so we actually have much more pivots (albeit local and with coarse data). Finally, the trees can be traversed with sublinear extra CPU time.

Different tree variants arise according to the tree arities, the way the ranges of distances are chosen (trying to balance the tree or not), how local are the pivots (different nodes can share pivots, which do not belong anymore to the subtree), the number of pivots per node, and so on. Very little is known about which is best. For example, the golden rule of preferring balanced trees, which works well for exact searching, becomes a poorer choice against unbalancing as the dimension increases. For very high dimensional data the best data structure is just a linked list (a degenerate tree) [3]. Also, little is known about how to choose the pivots.

## 4.2 Local Partitioning Schemes

Another scheme builds on the idea of dividing the database into spatially compact groups, meaning that the elements in each group are close to each other. A representative is chosen from each group, so that comparing $q$ against the representative has good chances of discarding the whole group without further comparisons. Usually these schemes are hierarchical, so that groups are recursively divided into subgroups.

Two main ways exist to define the groups. One can define "centers" with a covering radius, so that all elements in its group are within the covering radius distance to the center, e.g. [6]. If a group has center $c$ and covering radius $r_c$ then, if $d(q, c) > r + r_c$, it can be wholly discarded. The geometric shape of the above scheme corresponds to a ball around $c$. In high dimensions, all the balls tend to overlap and even a query with radius zero has to enter many balls.

This overlap problem can be largely alleviated with the second approach, e.g. [8]. The set of centers is chosen and every other point is added to the group of its closest center. At query time, if $q$ is closest to center $c_i$, and $d(q, c_j) - r > d(q, c_i) + r$, then we can discard the group of $c_j$. The geometric shape in this approach corresponds to a Dirichlet domain of the space (a generalization of the Voronoi diagram to metric spaces), without overlaps among groups.

The fact that balls can overlap has a good counterpart, however. A new point can be inserted inside any ball, enlarging the covering radius if necessary. With the Dirichlet domain, on the other hand, there is exactly one group where a new point must lie.

The flexibility at insertion is important when considering dynamism and secondary memory, since a flexible scheme permits filling disk pages better, balancing the tree if desired, and reshaping the tree at lower cost. In general, dynamism and secondary memory considerations have become a focus of attention only recently, and only few data structures consider them, e.g. [6]. This is an important issue to address in order to consider these structures for serious massive data management.

Finally, another important trend is that of probabilistic and approximate algorithms, e.g. [4]. These are especially welcome in this area because similarity searching usually means that there are no right and wrong answers, just better and worse ones. Hence it is acceptable to return suboptimal answers if this is done much faster. In particular, it has been shown that this type of techniques is a very powerful tool to cope with the dimensionality curse.

## Relevant Terms

**Curse of dimensionality:** the sharp dependency on the space dimension experimented by any search algorithm on vector or metric spaces.

**Histogram:** approximation, obtained by sampling, of a distribution (a distance distribution, for example).

**Index:** a data structure built on a database to speed up searches.

**Intrinsic difficulty:** a numerical measure of how hard searching a given database is, independently of the index used. It grows with the intrinsic dimension.

**Intrinsic dimension:** the minimum representational dimension where the database could be represented without distorting its distances.

**Metric database:** a database that provides access to a set of objects based on similarity to a query object, using the metric space model.

**Metric space:** a set of elements together with a distance function defined among pairs of elements.

**Distance:** a function from pairs of objects into nonnegative real numbers. It is zero only if the two arguments are the same. It must be symmetric and obey the triangle inequality.

**Multimedia data:** data that represent continuous objects of the real world (images, audio, etc.), searching which by exact equality is usually meaningless.

**Nearest neighbor query:** a similarity query that retrieves the closest elements to some query object.

**Metric Range query:** a similarity query that retrieves all elements within some distance to a query object.

**Representational dimension:** the number of coordinates a vector space has.

**Triangle inequality:** the property $d(x, y) \leq d(x, z) + d(z, y)$ that a distance $d$ must satisfy in order to be a metric.

## References

[1] C. Böhm, S. Berchtold, and D. Keim. Searching in high-dimensional spaces: Index structures for improving the performance of multimedia databases. *ACM Computing Surveys*, 33(3):322–373, 2001.

[2] E. Chávez, J.L. Marroquin, and G. Navarro. Fixed queries array: A fast and economical data structure for proximity searching. *Multimedia Tools and Applications (MTAP)*, 14(2):113–135, 2001.

[3] E. Chávez and G. Navarro. An effective clustering algorithm to index high dimensional met ric spaces. In *Proc. 7th Int'l Symp. on String Processing and Information Retrieval (SPIRE'2000)*, pages 75–86. IEEE CS Press, 2000.

[4] E. Chávez and G. Navarro. Probabilistic proximity search: Fighting the curse of dimensionality in metric spaces. *Information Processing Letters*, 85:39–46, 2003.

[5] E. Chávez, G. Navarro, R. Baeza-Yates, and J.L. Marroquin. Proximity searching in metric spaces. *ACM Computing Surveys*, 33(3):273–321, 2001.

[6] P. Ciaccia, M. Patella, and P. Zezula. M-tree: an efficient access method for similarity search in metric spaces. In *Proc. 23rd Conf. on Very Large Databases (VLDB'97)*, pages 426–435, 1997. Software available for download at `http://www-db.deis.unibo.it/Mtree/`.

[7] V. Gaede and O. Günther. Multidimensional access methods. *ACM Computing Surveys*, 30(2):170–231, 1998.

[8] G. Navarro. Searching in metric spaces by spatial approximation. *The Very Large Databases Journal (VLDBJ)*, 11(1):28–46, 2002.

[9] Peter N. Yianilos, Data Structures and Algorithms for Nearest Neighbor Search in General Metric Spaces Proceedings of the Fifth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), 1993