A Unified Model for Similarity Searching *

Edgar Chávez[1], Gonzalo Navarro[2], Ricardo Baeza-Yates[2], and José L. Marroquín[3]

[1] Univ. Michoacana, Morelia, Mich. México. elchavez@zeus.ccu.umich.mx.
[2] Dept. of Computer Science, Univ. of Chile, Santiago, Chile. {gnavarro,rbaeza}@dcc.uchile.cl.
[3] Cent. de Inv. en Mat. (CIMAT), Guanajuato, México. jlm@fractal.cimat.mx.

Abstract. The indexing algorithms and data structures for similarity searching in metric spaces seem to emerge from a great diversity, and different approaches have been proposed and analyzed separately, often under different assumptions. Currently, the only realistic way to compare two different algorithms is to apply them to the same data set. We present a unified model for studying similarity searching algorithms, defining common complexity measures allowing comparison between different approaches.

1 Introduction

A metric space is a set of "black box" objects on which a distance function has been defined. On this set one performs "similarity queries", whose simplest form is to have a new element q and a maximum distance r so that one wants to retrieve all the elements of the set at distance r or less from q. This is trivially solved by comparing q against every element in the set, but since the distance is expensive to compute, the goal is to structure the set so that the number of distance evaluations to answer the queries are minimized. This data structure built on the set is often called an "index".

This general problem has a lot of applications, such as non-traditional databases (where the concept of exact search is of no use and we search for similar objects, e.g. databases storing images, fingerprints or audio clips); machine learning and classification (where a new element must be classified according to its closest existing element); image quantization and compression (where only some vectors can be represented and those that cannot must be coded as their closest representable point); text retrieval (where we look for words in a text database allowing a small number of errors, or we look for documents which are similar to a given query or document); computational biology (where we want to find a DNA or protein sequence in a database allowing some errors due to typical variations); function prediction (where we want to search the most similar behavior of a function in the past so as to predict its probable future behavior); etc.

* Supported in by CYTED VII.13 AMYRI project, and also by CONACyT grant R-28923A (first author), Fondecyt grant 99-0627 (second and third authors) and CONACyT (fourth author).

Since the problem has appeared in unrelated areas, the corresponding algorithms and data structures seem to emerge from a great diversity, and different approaches have been proposed and analyzed separately, often under different assumptions [5, 20, 22, 19, 21, 23, 13, 15, 1, 4, 14, 18, 3, 11, 17, 7, 8, 24]. Due to space limitations we refer the reader to a recent survey where all the known approaches for similarity searching are discussed [9].

Currently, the only realistic way to compare two different algorithms is to apply them to the same data set. We present a unified complexity model for the search in metric spaces. Its main contribution can be summarized in Figure 1: all the indexing algorithms partition the set of elements into subsets. An index is built which allows to determine a set of candidate sets where the elements relevant for the query can appear. At query time, que index is searched to find the relevant subsets (the cost to do this is called "internal complexity") and those subsets are checked exhaustively (which corresponds to the "external complexity" of the search). There is a tradeoff between internal and external complexity: finer grained partitions have higher internal and lower external complexity.

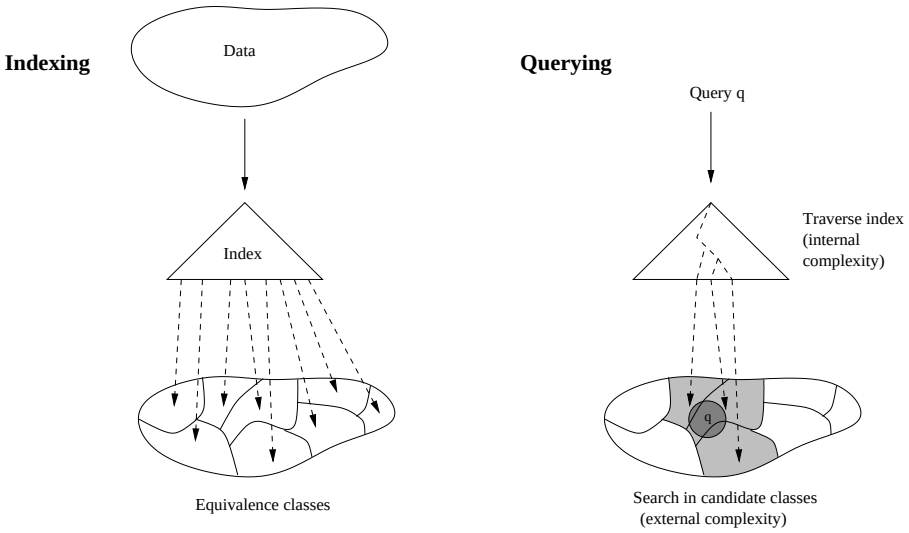

Fig. 1. The unified model for indexing and querying metric spaces.

2 Metric Spaces and Proximity Queries

We introduce now the basic notation for the problem. The set $\mathbb{X}$ will denote the universe of objects. A finite subset $\mathbb{U}$, of size $n = |\mathbb{U}|$, will be called the *dictionary*. The function $d : \mathbb{X} \times \mathbb{X} \longrightarrow \mathbb{R}$ will denote a measure of "similarity" between objects, satisfying the following properties for all $x, y, z \in \mathbb{X}$:

$(p1)$ $d(x,y) \geq 0$ positiveness,
$(p2)$ $d(x,y) = d(y,x)$ symmetry,
$(p3)$ $d(x,x) = 0$ reflexivity, and in most cases
$(p4)$ $d(x,y) = 0$ iff $x = y$ strict positiveness.

The similarity d will be called a **distance** if it satisfies the triangle inequality

$(p5)$ $d(x,y) \leq d(x,z) + d(z,y)$.

There are basically two types of queries of interest in metric spaces:

(a) Range queries: retrieve all elements which are within distance r to q. This is, $(q,r)_d = \{u \in \mathbb{U} \ / \ d(q,u) \leq r\}$.
(b) Nearest neighbor queries: retrieve the closest elements to q in $\mathbb{U}$. This is, $nn_d(q) = \{u \in \mathbb{U} \ / \ \forall v \in \mathbb{U},\ d(q,u) \leq d(q,v)\}$. We may also want the k nearest neighbors.

We restrict ourselves to the analysis of range queries, since nearest neighbor queries can be derived from range queries using a branch and bound scheme.

3 Equivalence Relations and Cosets

Given a set $\mathbb{X}$, a **partition** $\pi(\mathbb{X}) = \{\pi_1, \pi_2, \cdots, \pi_\ell, \cdots\}$ is a subset of the power set $\mathcal{P}(\mathbb{X})$ such that every element of the set belongs exactly to one partition, i.e.

$$\cup_1^\ell \pi_i = \mathbb{X}, \quad \text{and} \quad \pi_i \cap \pi_j = \phi \quad \text{for all } i \neq j$$

A relation, denoted by $\sim$, is a subset of the cross product $\mathbb{X} \times \mathbb{X}$ (the set of ordered pairs) of $\mathbb{X}$. Two elements x, y are said to be related, denoted by $x \sim y$, if the pair (x,y) is in the subset. A relation $\sim$ is said to be an **equivalence relation** if it satisfies the properties of

- $x \sim x$ for all $x \in \mathbb{X}$, reflexivity
- $x \sim y$ if and only if $y \sim x$, symmetry
- $x \sim y$ and $y \sim z$ then $x \sim z$, transitivity

It can be shown that every partition $\pi(\mathbb{X})$ induces an equivalence relation $\sim$ and, conversely, every equivalence relation induces a partition [10]. Two elements are related if they belong to the same partition. Every element π_i of the partition is then called an **equivalence class**. An equivalence class is often named after one of its representatives (any element of π_i can be taken as a representative). An alternative definition of an equivalence class of an element x is the set of all y such that $x \sim y$. We will denote the equivalence class of x as $[x] = \{y : x \sim y\}$.

Given the set $\mathbb{X}$ and an equivalence relation $\sim$, we obtain the quotient $\pi(\mathbb{X}) = \mathbb{X}/\sim$. It indicates the set of equivalence classes or cosets, obtained when applying the equivalence relation to the set $\mathbb{X}$.

For a fixed set $\mathbb{X}$, consider two equivalence relations $\sim_1$ and $\sim_2$. We say that $\sim_1$ is a **refinement** of $\sim_2$ if for any pair $x, y \in \mathbb{X}$ such that $x \sim_1 y$ then

necessarily $x \sim_2 y$. Equivalently, a partition $\pi^1(\mathbb{X})$ is a refinement of partition $\pi^2(\mathbb{X})$ if $\pi_i^1 \subset \pi_j^2$ for every partition element π_i^1 of π^1 and some coset π_j^2 of the partition π^2. We may also say that π^2 (equivalently $\sim_2$) is a **coarsening** of π^1 (equivalently $\sim_1$).

The relevance of equivalence classes for us comes from the possibility of using them on a metric space in a way that a new metric space is derived from the quotient set. This new metric space will be a coarser version of the original one.

4 Indexing and Partitions

The equivalence classes obtained with an equivalence relation of a metric space can be considered themselves as points in a new metric space, as soon as we define the distance function D of this new metric space.

In Figure 1 we can see a schematic example of the idea. We divide the space in several regions (equivalence classes). The points inside each region are indistinguishable. We consider them as points in a new metric space. To answer a query, we first search the relevant classes in the quotient space. Then, instead of exhaustively examining the entire dictionary we just examine the classes that contain potentially interesting points. In other words, if a class can contain a point that should be returned in the outcome of the query, then the class will be examined.

For this approach to be useful we need that the mapping is *contractive*, i.e. $D([a],[b]) \leq d(a,b)$ for any $a, b \in \mathbb{X}$. This ensures that all the points relevant to a query $(q,r)_d$ are contained in the classes returned by the query $([q],r)_D$ on the quotient space. We explain the idea in more detail now.

We introduce a new function $D_0 : \pi(\mathbb{X}) \times \pi(\mathbb{X}) \longrightarrow \mathbb{R}^+$ now defined in the quotient. Since D_0 is defined between equivalence classes, a natural choice is $D_0([x],[y]) = \inf_{x \in [x], y \in [y]}\{d(x,y)\}$, which we call the **extension** of d. We define the outcome of a query in the coset as $([q],r)_{D_0} = \{u \in \mathbb{U} : D_0([u],[q]) \leq r\}$.

D_0 could be used for similarity searching in a natural way. From the definition it is clear that $D_0([x],[y]) \leq d(x,y)$ for any $x \in [x]$, $y \in [y]$. This permits to convert one search problem into another, hopefully simpler, search problem. For a given query $(q,r)_d$ we find out the equivalence class $[q]$ the query point q belongs to. Then, using the new distance function D_0 the query is transformed into $([q],r)_{D_0}$. Since $D_0([q],[u]) \leq d(q,u)$, this naturally implies $(q,r)_d \subseteq ([q],r)_{D_0}$. That is, $([q],r)_{D_0}$ is indeed a list of candidates, so it is enough to perform an exhaustive search on that candidate list (now using the original distance d), to obtain the actual outcome of the query $(q,r)_d$.

Unfortunately, D_0 does not satisfy the triangle inequality, just $(p1)$ to $(p3)$, and in most cases (p_4). Hence, no general search algorithm is possible using D_0 itself. However, D_0 gives the maximum possible values that keep the mapping contractive, and therefore we can use any other distance that satisfies the properties of metric spaces and that lower bounds D_0. This distance, D, can be used for indexing purposes and still serves to obtain a list of candidates for the actual

outcome using d. An example of D is given in Figure 2, where the equivalence classes are rings centered around a point p.

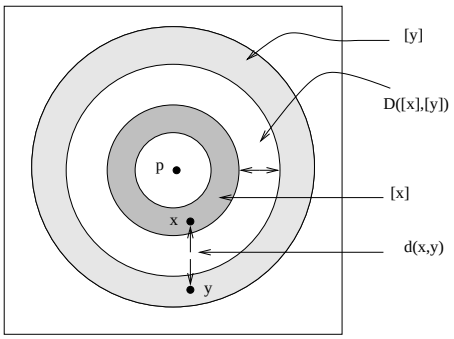

Fig. 2. Two points x and y, and their equivalence classes (the shaded rings). D gives the minimum distance among rings, which lower bounds the distance between x and y.

When selecting D, another important consideration is the cost to compute it. In fact, the most important tradeoff for an indexing algorithm is to keep low the number of evaluations to compute the D distance, and at the same time to reduce the final exhaustive search. In fact, the above procedure is used in virtually every indexing algorithm. In other words:

> *Most indexing algorithms for proximity searching consist in building a set of equivalence classes, discard some classes, and search exhaustively the rest.*

Some examples may help to understand the above definitions.

Example 1. The brute force method of not indexing and examining every point in the dictionary for each query, creates one equivalence class per point in the set $\mathbb{X}$. In this case, the coset obtained is the same as the original set $\pi(\mathbb{X}) = \mathbb{X}/\sim = \mathbb{X}$, it holds $x \sim y \iff x = y$, $D_0([x],[y]) = d(x,y)$ for any pair x, y and consequently $(q,r)_d = ([q],r)_{D_0}$. In other words the candidate list is actually the outcome of the query. No extra effort is done in trimming the candidate list, however all the work have been done in building the candidate list.

Example 2. Another trivial example, situated in the other side of the spectrum, is when all points in $\mathbb{X}$ are assigned to the same equivalence class, i.e. $x \sim y \iff x, y \in \mathbb{X}$. Hence $[x] = [y]$ for all elements in the set $\mathbb{X}$. In this case we have $\pi(\mathbb{X}) = \mathbb{X}/\sim = \{\cdot\}$, a set with a single element, and it holds $D_0([x],[y]) = 0$ for every pair of points. In this case finding the candidate list is trivial, since it is actually the dictionary itself, but trimming the list is as difficult as the original problem.

Example 3. A more realistic example, indeed a true indexing algorithm, is when we have an arbitrary reference point $p \in \mathbb{X}$ and the equivalence relation is given by $x \sim y \iff d(p,x) = d(p,y)$. In this case $D([x],[y]) = |d(x,p) - d(y,p)|$ is a safe lower bound for the D_0 distance (guaranteed by triangle inequality). For a query of the form $(q,r)_d$ the candidate list $([q],r)_D$ consist of all points such that $D([q],[x]) \leq r$ or in other way all the points such that $|d(q,p) - d(x,p)| \leq r$. Graphically, this distance represents a *ring* centered at p containing a *disk* centered at q and radius r (recall Figure 2). This is the familiar rule used in many independent algorithms to trim the space, as seen later.

Example 4. The approximate search problem was firstly introduced in "vector spaces" ($\mathbb{R}^k$), and the very first family of algorithms used there were based on a coset operation. These were called *bucketing* methods, and consist in the construction of cells or buckets [2]. Searching for an arbitrary point in $\mathbb{R}^k$ is converted into an exhaustive search in a finite set of cells. The procedure used two steps: (1) first they find which cell the query point belongs to and after that they build a set of candidate cells using the query range; (2) this set of candidate cells is inspected exhaustively to find the actual points inside the query range. In this case the equivalence classes are the cells, and the tradeoff is expressed as follows: the larger the cells, the cheaper it is to find the appropriate ones, but the more costly is the final exhaustive search.

5 Measures of Efficiency

As sketched previously, most indexing algorithms rely on building an equivalence class. The corresponding search algorithms have two parts:

1. Find the classes that may be relevant for the query.
2. Exhaustively search all the elements of these classes.

The first part involves performing some evaluations of the d distance, as shown in the Example 3 above. It may also involve some extra CPU time (which although not the central point in this paper, must be kept reasonable). The distance evaluations performed in this stage are called **internal**, and their number define the *internal complexity*.

The second part consists of directly comparing the query against the candidate list. These evaluations of d are called **external**. The amount of external evaluations is called *external complexity* and is related to the **discriminative power** of the D distance, a concept that we explain shortly.

The indexing scheme needs to find a balance between the complexity to compute D and its discriminative power.

Examples 1 and 2 can serve as upper and lower bounds of what is done by the actual indexing algorithms. The first algorithm has minimal external complexity, since the distance function D discriminates as much as the original distance function d. However, the internal complexity is maximal, in the sense that finding the relevant classes is as hard as solving the original problem. This case shows

maximum discriminative power, as the metric spaces $(\mathbb{X}, d)$ and $(\pi(\mathbb{X}), D)$ are isometric [16]. Example 2 has minimal internal complexity, since it is trivial to compute the relevant equivalence class. However, its external complexity is as high as in the original problem, since all the points are candidates.

Example 3 is in between for internal and external complexity. The internal complexity is 1 distance evaluation (the distance from q to p), and the external complexity will correspond to the number of elements that lie in the selected ring. We could intersect it with more rings (increasing internal complexity) to reduce the external complexity.

The tradeoff is partially formalized with the notions of refinement and coarsening of a partition. In particular, the following theorems show that the external complexity decreases as the partition is more refined (and we may assume that the internal complexity increases since more information has to be obtained).

Theorem 1. *If $\sim_1$ is a coarsening of $\sim_2$ then the extended distances D_1 and D_2 have the property $D_2([x],[y]) \leq D_1([x],[y])$.*

Proof. $D_1([x],[y]) = \inf_{x \in [x]_1, y \in [y]_1}\{d(x,y)\} \leq \inf_{x \in [x]_2, y \in [y]_2}\{d(x,y)\} = D_2([x],[y])$, since $[x]_2 \subseteq [x]_1$ and $[y]_2 \subseteq [y]_1$. We are using $[x]_i$ and $[y]_i$ to denote the equivalence class of x and y under equivalence relation $\sim_i$.

Theorem 2. *If A_1 and A_2 are indexing algorithms based on equivalence relations $\sim_1$ and $\sim_2$, respectively, and $\sim_2$ is a coarsening of $\sim_1$, then A_1 has lower external complexity than A_2.*

Proof. We have to show that $([q],r)_{D_1} \subseteq ([q],r)_{D_2}$. But this is clear, since $D_2([x],[y]) \leq D_1([x],[y])$ implies $([q],r)_{D_1} = \{y \in \mathbb{U} : D_1([q],[y]) \leq r\} \subseteq \{y \in \mathbb{U} : D_2([q],[y]) \leq r\} = ([q],r)_{D_2}$.

An interesting idea arising from the above theorems is to build a hierarchy of coarsening operations. Using this hierarchy we could proceed downwards from a very coarse level building a candidate list of equivalence classes of the next level, using for example D^j; this candidate list will be refined using the D^{j-1} distance function and so on until we reach the bottom level. This is done, e.g. in [4].

The concept of discriminative power serves as an indicator of the performance or fitness of the equivalence relation (or equivalently, of the distance function D). In general, it will be more costly to have more discriminative power. A related concept is that of "fragmentation", which is explained next.

5.1 Locality and Fragmentation of a Partition

The equivalence classes can be thought of as a set of *non intersecting* cells in the space, where every point inside a given cell belongs to the same equivalence class. Of course, this is not very precise from a mathematical point of view, but can serve to clarify the concept. In the mathematical definition an equivalence class is *not* confined to a single cell; indeed it can be an arbitrary collection of cells.

A consequence of the above observation is that we need an additional property which will be called **locality**. It stands for how much the equivalence class resembles a cell. Intuitively the opposite property, **fragmentation** of a partition, can be easier to understand. An equivalence class is *fragmented* if it consists of several "local" pieces. We do not go beyond with the formalization of this concept, because the definition involves properties not common to every metric space and we want to maintain our model as general as possible.

Figure 3 exemplifies a fragmented partition. It is natural to expect a better performance, i.e. more discriminative power, from a local partition than from a fragmented partition. This is because the candidate list obtained with the distance D will contain points actually far away from the query if the partition is fragmented. Notice that in Figure 3, the fragmentation would disappear if we added a third pivot.

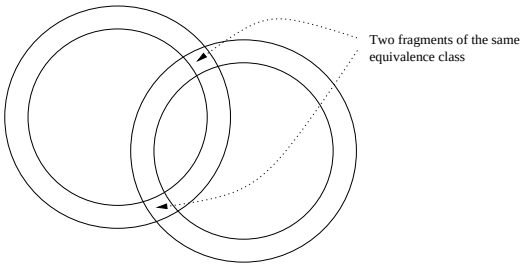

Fig. 3. With two rings we define an equivalence based on being at the same distance to both points. However, the resulting class is partitioned.

6 Pivot Based and Clustering Algorithms

A large class of methods to index metric spaces are just variants of what we call "pivot-based algorithms" [5, 20, 22, 21, 23, 13, 15, 1, 14, 18, 3, 11, 7, 8, 24]. The idea is an extension of Example 3, using more pivots in order to decrease the external complexity. Instead of just one pivot, one selects h "pivots" $p_1 \cdots p_h \in \mathbb{U}$, and stores all the distances $d(u, p_i)$ for all $u \in \mathbb{U}$. This set of distances is the index. Now, given a query $(q, r)_d$, q is compared against each pivot. This costs h distance evaluations, which is the internal complexity.

The elements are filtered out with the following reasoning. For any $u \in \mathbb{U}$, the distance $d(q, u)$ cannot be smaller than $|d(u, p_i) - d(q, p_i)|$ for any pivot p_i, because of the triangular inequality. Therefore, all the elements u such that $|d(u, p_i) - d(q, p_i)| > r$ for some p_i can be safely discarded (recall that $d(u, p_i)$ is precomputed). The external complexity then corresponds to comparing q directly against all those $u \in \mathbb{U}$ that could not be discarded.

In terms of our model, we have that the equivalence classes correspond to the intersection of sphere shells centered at the p_i's (recall Figure 3). Hence

$D([x],[y]) = \max_i\{|d(x,p_i) - d(y,p_i)|\}$ is a safe lower bound to the D_0 function corresponding to these equivalence classes.

As explained, the internal complexity increases and the external complexity decreases as h grows, so there exists an optimum. We present an experiment showing this phenomenon. Our metric space is the unitary real cube in k dimensions ($[0,1)^k$) under the Euclidean distance. We generated $n = 100,000$ random points and search a random query q with a radius r such that 0.01% of the dictionary is retrieved.

Figure 4 (left) shows internal, external and total distance evaluations in 8 dimensions, using up to 256 pivots. The optimum number of pivots for this case is close to $h = 110$. Despite that with 256 pivots we reach an optimum only until 8 dimensions, it seems clear that the optimal h grows very fast with k. The optimal h is 15, 30 and 110 for 4, 6, and 8 dimensions, respectively. Figure 4 (right) shows the overall complexity in higher dimensions, where the optimum is not achieved with 256 pivots.

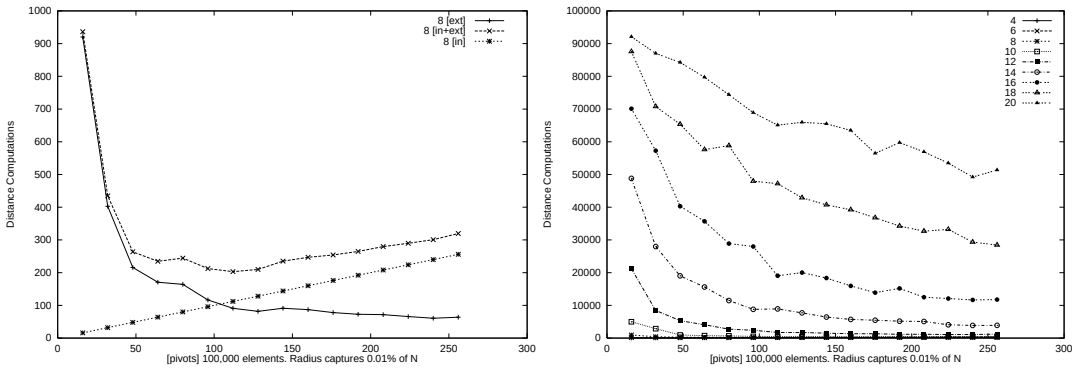

Fig. 4. On the left, internal, external and overall distance evaluations in 8 dimensions, using different number of pivots. On the right, overall distance evaluations for different number of dimensions.

This also shows that a lot of memory is required for high-dimensional spaces, because of which most indexing algorithms do not reach this optimum but they us as many pivots as memory permits.

Another kind of algorithms, less popular, is called "clustering" algorithms [5, 4, 17]. The idea is also partitioning the space in equivalence classes, but the classes are not defined as being at the same distance to a set of pivots. In this case the classes are normally being closer to a pivot than to any other pivot. The result is that the classes are more local (less fragmented). On the other hand, clustering indices tend to have high construction costs and cannot be easily improved by adding more and more pivots.

7 The Curse of Dimensionality and Conclusions

The fact that the optimum number of pivots grows quickly with the dimension is just one aspect of the general phenomenon called "curse of dimensionality". It is expressed as an exponential dependency on k that appears in the performance of *any* algorithm to index k-dimensional vector spaces. It is interesting that the phenomenon appears also when we disregard the coordinates and view the vector space just as a metric space, so that one can in general speak of "high-dimensional" metric spaces even when no explicit concept of dimension exists.

This phenomenon has a general cause which has also been mentioned in [23, 4, 6, 12, 9]. Very briefly, high dimensional metric spaces have a more concentrated histogram of distances, and therefore a random query selecting a band of width $2r$ of the histogram of distances to a pivot p_i captures much more points in high dimensional metric spaces. Only the points outside the band can be discarded, and therefore more and more pivots have to be considered to discard a significative amount of points. Figure 5 illustrates.

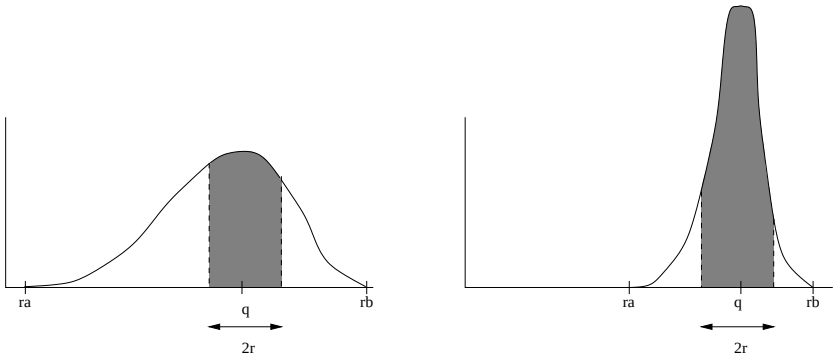

Fig. 5. A low-dimensional (left) and high-dimensional (right) histogram of distances, showing that on high dimensions virtually all the elements become candidates for the exhaustive evaluation.

Another facet of high dimensionality is the difficulty of obtaining local partitions. Even with perfect pivot selection we need $k+1$ points to obtain a local partition in $\mathbb{R}^k$ (recall Figure 3). In general metric spaces one resorts to random pivot selection and this number is much higher. This shows why clustering algorithms may be better suited for high dimensional spaces, as the experiments in [9] show: fragmentation is smaller in the equivalence relations produced by clustering algorithms. As a consequence, pivot based algorithms need much more memory than clustering algorithms to beat them in high dimensions.

The most interesting research line on indexing metric spaces seems to be the development of a clustering approach that can be improved by using more memory and that needs less memory than pivot based algorithms to achieve the same performance.

References

1. R. Baeza-Yates, W. Cunto, U. Manber, and S. Wu. Proximity matching using fixed-queries trees. In *Proc. 5th Combinatorial Pattern Matching (CPM'94)*, LNCS 807, pages 198–212, 1994.
2. J. Bentley, B. Weide, and A. Yao. Optimal expected-time algorithms for closest point problems. *ACM Trans. on Mathematical Software*, 6(4):563–580, December 1980.
3. T. Bozkaya and M. Ozsoyoglu. Distance-based indexing for high-dimensional metric spaces. In *Proc. ACM SIGMOD International Conference on Management of Data*, pages 357–368, 1997. Sigmod Record 26(2).
4. S. Brin. Near neighbor search in large metric spaces. In *Proc. 21st Conference on Very Large Databases (VLDB'95)*, pages 574–584, 1995.
5. W. Burkhard and R. Keller. Some approaches to best-match file searching. *Comm. of the ACM*, 16(4):230–236, April 1973.
6. E. Chávez and J. Marroquín. Proximity queries in metric spaces. In R. Baeza-Yates, editor, *Proc. 4th South American Workshop on String Processing (WSP'97)*, pages 21–36. Carleton University Press, 1997.
7. E. Chávez, J. Marroquín, and R. Baeza-Yates. Spaghettis: an array based algorithm for similarity queries in metric spaces. In *Proc. String Processing and Information Retrieval (SPIRE'99)*, Cancun, Mexico, September 1999. To appear. `ftp://garota.fismat.umich.mx/pub/users/elchavez/spa.ps.gz`.
8. E. Chávez, J. Marroquín, and G. Navarro. Overcoming the curse of dimensionality. In *European Workshop on Content-Based Multimedia Indexing (CBMI'99)*, Tolouse, France, October 1999. To appear. `ftp://garota.fismat.umich.mx/-pub/users/elchavez/fqa.ps.gz`.
9. E. Chávez, G. Navarro, R. Baeza-Yates, and J. Marroquín. Searching in metric spaces. Technical Report TR/DCC-99-3, Dept. of Computer Science, Univ. of Chile, 1999. `ftp://ftp.dcc.uchile.cl/pub/users/gnavarro/-survmetric.ps.gz`.
10. L. Childs. *A Concrete Introduction to Higher Algebra*. Springer-Verlag, 1995.
11. P. Ciaccia, M. Patella, and P. Zezula. M-tree: an efficient access method for similarity search in metric spaces. In *Proc. of the 23rd Conference on Very Large Databases (VLDB'97)*, pages 426–435, 1997.
12. P. Ciaccia, M. Patella, and P. Zezula. A cost model for similarity queries in metric spaces. In *Proc. 17th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS'98)*, 1998.
13. A. Faragó, T. Linder, and G. Lugosi. Fast nearest-neighbor search in dissimilarity spaces. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 15(9):957–962, September 1993.
14. L. Micó, J. Oncina, and R. Carrasco. A fast branch and bound nearest neighbour classifier in metric spaces. *Pattern Recognition Letters*, 17:731–739, 1996.
15. L. Micó, J. Oncina, and E. Vidal. A new version of the nearest-neighbor approximating and eliminating search (AESA) with linear preprocessing-time and memory requirements. *Pattern Recognition Letters*, 15:9–17, 1994.
16. J. Munkres. *Topology, A First Course*. Prentice-Hall, 1975.
17. G. Navarro. Searching in metric spaces by spatial approximation. In *Proc. String Processing and Information Retrieval (SPIRE'99)*, Cancun, Mexico, September 1999. To appear. `ftp://ftp.dcc.uchile.cl/pub/users/gnavarro/-metric.ps.gz`.

18. S. Nene and S. Nayar. A simple algorithm for nearest neighbor search in high dimensions. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 19(9):989–1003, 1997.
19. D. Sasha and T. Wang. New techniques for best-match retrieval. *ACM Trans. on Information Systems*, 8(2):140–158, 1990.
20. M. Shapiro. The choice of reference points in best-match file searching. *Comm. of the ACM*, 20(5):339–343, May 1977.
21. J. Uhlmann. Satisfying general proximity/similarity queries with metric trees. *Information Processing Letters*, 40:175–179, 1991.
22. E. Vidal. An algorithm for finding nearest neighbors in (approximately) constant average time. *Pattern Recognition Letters*, 4:145–157, 1986.
23. P. Yianilos. Data structures and algorithms for nearest neighbor search in general metric spaces. In *Proc. 4th ACM-SIAM Symposium on Discrete Algorithms (SODA'93)*, pages 311–321, 1993.
24. P. Yianilos. Excluded middle vantage point forests for nearest neighbor search. In *DIMACS Implementation Challenge, ALENEX'99*, Baltimore, MD, 1999.