The Basic Principles of Metric Indexing

Magnus Lie Hetland

Norwegian University of Science and Technology

Summary. This chapter describes several methods of similarity search, based on metric indexing, in terms of their common, underlying principles. Several approaches to creating lower bounds using the metric axioms are discussed, such as pivoting and compact partitioning with metric ball regions and generalized hyperplanes. Finally, pointers are given for further exploration of the subject, including non-metric, approximate, and parallel methods.

⟨**comment**⟩ In this draft, notes are typeset inline. Other items tagged in a similar fashion (such as this paragraph) are omitted in the final document. A continuation symbol ('↪') means that the text is a continuation of the previous untagged paragraph; any intervening paragraph breaks should be ignored. ⟨/**comment**⟩

Introduction

This chapter is a tutorial – a brief introduction to the field of metric indexing, which is a way of optimizing certain kinds of similarity retrieval. While there are several excellent publications that cover this area, this chapter takes a slightly different approach in several respects.

⟨**note**$_1$⟩ Chávez et al. [33] have written a very thorough and rather theoretical survey paper on the subject. In addition to describing the methods available at the time, they construct a taxonomy of the methods and derive theoretical bounds for their complexity. They also introduce the concept of intrinsic dimensionality (see Sect. 7). The later survey by Hjaltason and Samet [3] takes a somewhat different approach, and uses a different fundamental formalism, but is also quite comprehensive and solid, and complements the paper by Chávez et al. nicely. In recent years, a textbook on the subject by Zezula et al. has appeared [39], and Sect. 4.5 of Samet's book on multidimensional and metric data structures [4] is also devoted to distance based methods. The paper by Pestov and Stojmirović [96] is specifically about a model for similarity search, but in many ways provides a brief survey as well. The encyclopedia entry by Chávez and Navarro [97] is another example of a brief introduction. In addition to publications that specifically set out to describe the field, there are also publications, such as the PhD thesis of Skopal [98], with more specific topics, that still have substantial sections devoted to the general field of metric indexing. ⟨/**note**$_1$⟩

↪ Primarily, it focuses on giving a concise explanation of underlying principles rather than a comprehensive survey of specific methods (which means that some methods are explained in ways that differ significantly from the original publications). Also, the main effort has been put into explaining these principles clearly, rather than going in depth theoretically or covering the full breadth of the field.

The first two sections of this chapter provide an overview of the main goals and principles of similarity retrieval in general. Section 3 discusses the specifics of metric spaces, and the following sections deal with three approaches to metric indexing: pivoting, ball partitioning, and generalized hyperplane partitioning. Section 7 summarizes some methods and issues that aren not dealt with in detail elsewhere, and finally the appendices give some mathematical details, as well as a listing of all the specific indexing methods discussed in the chapter. To enhance the flow of the text, many technical details have been relegated to end notes, which can be found on pp. 31–31.

1 The Goals of Distance Indexing

Similarity search is a mode of information retrieval where the query is a sample object, and the desired result is a set of objects deemed to be similar – in some sense – to the query. The similarity is usually formalized (inversely) as a so-called *distance function*,* and indexing is any form of preprocessing of the data set designed to facilitate efficient retrieval.

⟨**note**$_2$⟩ Note that the terminology is not entirely consistent across the literature. The use of 'distance' here conforms with the usage of Deza and Deza [99]. ⟨/**note**$_2$⟩

*A distance function (or simply a *distance*) d is a non-negative, real-valued, binary function that is reflexive ($d(x,x)=0$) and symmetric ($d(x,y)=d(y,x)$). The function is defined over some universe of possible objects, $\mathbb{U}$ (that is, it has the signature $d : \mathbb{U}^2 \rightarrow \mathbb{R}_0^+$). Both here and later on in the chapter, constraints are implicitly quantified over $\mathbb{U}$; for example, symmetry implies that $d(x,y)=d(y,x)$ *for all objects* x,y in $\mathbb{U}$. When discussing queries, it is assumed that query objects may be arbitrarily chosen from $\mathbb{U}$, while the returned objects are taken from some finite subset $\mathbb{D} \subseteq \mathbb{U}$ (the *data set*).

$\hookrightarrow$ The applications range from entertainment and multimedia (such as image or music search) to science and medicine (such as data mining or matching biological sequences), or anything else that requires efficient query-by-example, but where traditional (coordinate-based) spatial access methods cannot be used. Beyond direct search, similarity retrieval can be used as an internal component in a wide spectrum of systems, such as nearest neighbor classification (with large sample sets), compressed video streaming (to reuse image patches) and multiobjective optimization (to avoid near-duplication of solutions).

$\langle$**note**$_3\rangle$ Many of the surveys mentioned previously [3, 4, 33, 39, 96–98] discuss various applications, as do most publications about specific metric indexing methods. Spatial access methods [see, e.g., 4] can also be used for many query-by-example applications, but they rely on the specific structure of the problem – the fact that all objects are represented as vectors of a space of fixed, low dimensionality. Metric indexing is designed for cases where less is known about the space (i.e., where we only know the distances between objects, and those distances satisfy the metric axioms), and thus have a different, possibly wider, field of applications. Because the metric indexing methods disregard the number of dimensions of a vector space and are only hampered by the innate complexity of the given distance (or the distribution of objects), they may also be better suited to indexing high-dimensional traditional vector spaces (see Sect. 7). $\langle$/**note**$_3\rangle$

Under the distance function formalism, several query types may be formulated. In the following I focus on one of the basic kinds, so-called *range queries*, where all objects that fall within a given distance of the sample are returned. In other words, for a distance function d, a query object q, and a *search radius* r, objects x for which $d(q, x) \leq r$ are returned (see Fig. 1 on the following page). While alternatives are discussed in the literature (most notably returning the nearest object, or the k nearest objects) it can be argued that range queries are fundamental, and virtually all published metric indexing methods support them. (For more information about other query types, see Sect. 7.)

Index structures (such as the inverted files traditionally used in text search) are structures built over the given data set in order to speed up queries. The time cost involved in building the index is amortized over the series of queries, and is usually ignored when considering search cost. The main goal, then, of an index method is to enable efficient search, either asymptotically or simply in real wall-clock time. However, in any specialized form of search there may be a few wrinkles to the story; for example, in examining an index structure theoretically, some basic operations of the search algorithm may completely dominate others, but it may not be entirely clear which ones are the more costly, and there may be different constraints (such as memory use) or required pieces of functionality (such as being able to accomodate new objects) that

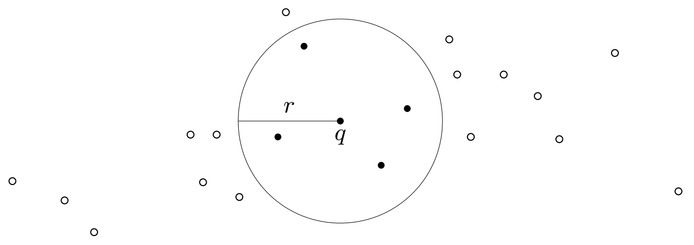


Fig. 1. Visualization of a range query in two-dimensional Euclidean space. The small circles are objects in the data set, while the filled dots are returned as a result of the query with sample object q and search radius r

influence what the criteria of optimization are. The three most important measures of quality used in the literature are:

- The number of distance computation needed during a query;
- The number of I/O operations (disk accesses) needed; and
- The CPU time used beyond distance computations or I/O operations.

Of these three, the first is of primary importance, mainly because it is generally assumed that the distance computations (which may involve comparing highly complex objects, such as video clips) are expensive enough to completely dominate the running time. Beyond this, some (mostly more recent) methods provide mechanisms for going beyond main memory without incurring an inordinate number of disk accesses, and many methods also involve "tricks" for cutting down on the CPU time. It is quite clear, though, that an underlying assumption for most of the field is that minimizing the number of distance computation is the main goal.

⟨**note**$_4$⟩ The field in question is defined by in excess of fifty published methods for metric indexing, stretching back to the 1983 paper of Kalantari and McDonald [45] (with more recent publications including ones by Aronovich and Spiegler [47] and Brisaboa et al. [90], for example), not counting several methods aimed specifically at discrete distance measures (surveyed along with more general methods by Chávez et al. [33]; see also Sect. 7). ⟨/**note**$_4$⟩

2 Domination, Signatures, and Aggregation

Without even considering the specifics of metric spaces, it is possible to outline some mechanisms that can be of use when implementing distance indices. This section discusses some fundamental principles, which may then be implemented, so to speak, using metric properties (dealt with in the following section), as shown in sections 4 through 6.

The most basic algorithm for similarity retrieval (or any form of search) is the *linear scan*: Traverse the entire data set, examining each object for eligibility.

⟨**note**$_5$⟩ While a linear scan is very straightforward, and seemingly quite inefficient, it may serve as an important "reality check," particularly for complex index structures (especially involving disk access) or high-dimensional data. The extra work in maintaining and traversing the structures may become so high, in some cases, that it swamps the reduction in distance computations [see, e.g., 72]. ⟨/**note**$_5$⟩

↪ In order to improve upon the linear scan, we must somehow infer that an object x can be included in, or excluded from, the search result *without* calculating $d(q, x)$. The way to go is to find a cheap way of *approximating* the distance. In order to avoid wrongfully including or excluding objects, we need an approximation with certain properties.

⟨**note**$_6$⟩ As discussed in Sect. 7, there are approximate retrieval methods as well, where some such errors are permitted. In these cases, looser notions of distance approximation may be acceptable. ⟨/**note**$_6$⟩

↪ In particular, the approximated distance must either overestimate or underestimate the actual distance, and must do so consistently. If the distance function $\hat{d}$ consistently yields higher values than another distance function $\check{d}$ (over the same set of objects), we say that $\hat{d}$ *dominates* $\check{d}$.

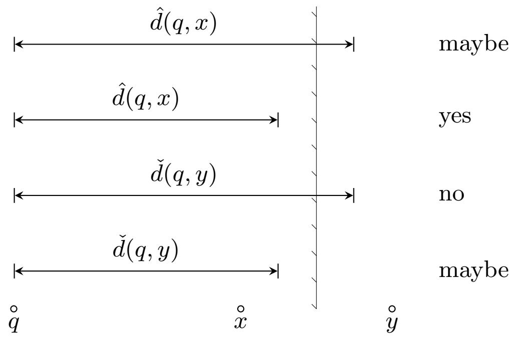


Fig. 2. Filtering through distance estimates. The vertical hatched line represents the range of the query; if an overestimated distance ($\hat{d}$) falls within the range, the object is included, and if an underestimated distance ($\check{d}$) falls beyond it, the object is discarded. Otherwise, the true distance must be computed

Let $\hat{d}$ be a distance that dominates the actual distance d (i.e., it forms an upper bound to d), and let d dominate another distance $\check{d}$ (a lower bound).

These estimates may then be used to partition the data set into three categories: no, yes, and maybe. If $\check{d}(q, o) > r$, the object o may be safely excluded (because no objects may, under $\check{d}$, be "closer than they appear"). Conversely, if $\hat{d}(q, o) \leq r$, the object o may be safely included. The actual distance $d(q, o)$ must then be calculated for any objects that do not satisfy either criterion. See Fig. 2 on the previous page and Fig. 3 on this page for illustrations of this principle. The more closely $\hat{d}$ and $\check{d}$ approximate d, the smaller the maybe set will be; however, there will normally be a tradeoff between approximation quality and cost of computation. One example of this kind of tradeoff is when there are several sources of knowledge available, and thus several upper and lower bounds; these may easily be combined, by letting $\hat{d}$ be the minimum of the upper bounds and $\check{d}$, the maximum of the lower bounds.

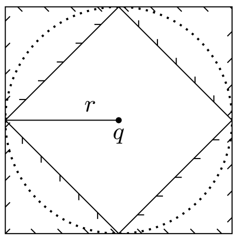


Fig. 3. An example of filtering through domination, where $\hat{d}$, d, and $\check{d}$ are all Minkowski metrics, $L_p(x, y) = \sqrt[p]{\sum_i |x_i - y_i|^p}$, with $p = 1$, 2, and ∞, respectively. The circle is the set of points x for which $d(q, x) = r$. The outer square and the inner diamond represent similar regions for $\check{d}$ and $\hat{d}$. Objects inside the diamond are safe to include, objects outside the square may be safely excluded, while the region between them (indicated by the hatching) contains the "maybe" objects

Note that under the (quite reasonable) assumption that most of the data set will *not* be returned in most queries, the notion of lower bounds and automatic exclusion is more important than that of upper bounds and automatic inclusion. Thus, a major theme in the quest for better distance indexing structures is the search for ever more accurate, yet still cheap, lower-bounding estimates.

One way of viewing such lower bounds is in terms of so-called *non-expansive mappings*: All objects are mapped to new objects, or *signatures*, and the *signature distance* becomes an underestimate for the original distance (see Fig. 4 on the next page).

$\langle$**note**$_7\rangle$ Such mappings are normally defined between metric spaces, and are known by several names, including *metric mappings* and 1-*Lipschitz mappings*. $\langle/$**note**$_7\rangle$

$\hookrightarrow$ If one has knowledge about the internal structures of the objects, such signature spaces may be defined quite specifically using this domain knowledge.

⟨**note**$_8$⟩ One example of this technique is similarity indexing for time series, where many signature types have been suggested, normally in the form of fixed-dimensional vectors that may be indexed using spatial access methods. Hetland [100] gives a survey of this application. ⟨/**note**$_8$⟩

↪ In the general case of distance indexing, however, this is not possible, and the geometry of the space itself must be exploited. For metric spaces, certain applications of pivoting (see Sect. 4) make the so-called pivot space quite explicit.

$$\begin{tikzcd}
x \arrow[r, |<->|, "d"] \arrow[d, "f"'] & y \arrow[d, "f"] \\
f(x) \arrow[r, |<->|, "\check{d}"'] & f(y)
\end{tikzcd}$$

Fig. 4. The non-expansive mapping f maps objects from a a universe of objects, $\mathbb{U}$, to a signature space, $\mathbb{P}$. The distance d is defined on $\mathbb{U}$, while $\check{d}$ is defined on $\mathbb{P}$. For all x, y in $\mathbb{U}$, we have that $d(x, y) \geq \check{d}(f(x), f(y))$

Many indexing methods add another mechanism to the (possibly signature-based) bounds, namely *aggregation*: The search space is partitioned into regions, and all objects within a region are handled collectively. At the level of object filtering, this does not give us any obvious advantages. In order to find a common lower bound that lets us discard an entire region, the bound must be defined as the minimum over all possible objects in that region – clearly not an improvement in accuracy. (Another take on this lower bound is to see it as a non-expansive mapping to $\mathbb{R}$, where no objects in the region have positive signatures; see Fig. 5 on the following page.)

⟨**note**$_9$⟩ Hjaltason and Samet [3] use hierarchical decompositions of the space into regions (represented as tree structures) as their main formalism when discussing indexing methods. In order to discard such a region R from a search, a lower bound on the point-set distance $d(q, R) = \inf_{x \in R} d(q, x)$ must be defined; this bound is a characteristic of the region type used. Pestov and Stojmirović [96] use basically the same formalism, although presented rather differently. Instead of equipping the tree nodes corresponding to regions with lower-bounding distance estimates, they give them *certification functions*, which are non-expansive (1-Lipschitz) mappings $f : R \to \mathbb{R}$, where $f(x) \leq 0, \forall x \in R$. While not discussed by the authors, it should be clear that these certification functions are equivalent to lower-bounding estimates of the point-set distance (see Fig. 4 on the previous page). For a rather different approach, see the survey by Chávez et al. [33]. They base their indexing formalism on the hierarchical decomposition of the search space into equivalence classes, and discuss overlapping regions without directly involving lower bounds in the fundamental model. ⟨/**note**$_9$⟩

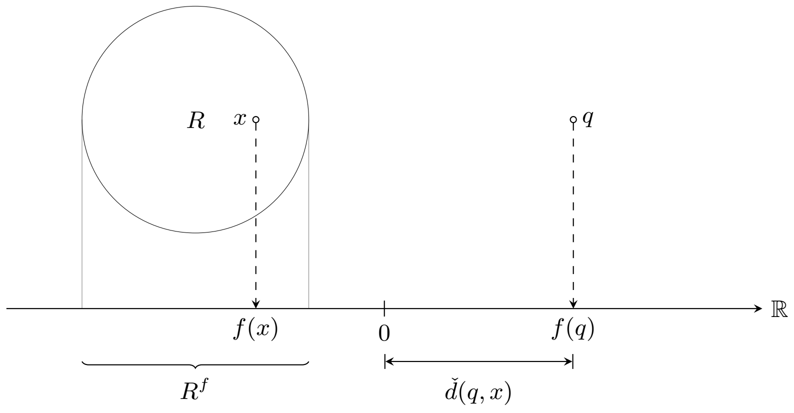


Fig. 5. Illustration of a region R with a non-expansive *certification function* (that is, a mapping to the real numbers, $\mathbb{R}$), where $f(x) \leq 0$ for all x in R. R^f is the image of R, mapped to $\mathbb{R}$ through f. As can be seen in this example, because R^f is required to lie below zero, $\check{d}(q,x) = f(q) \leq d(q,x)$, for all x in R. Note that if $f(q) \leq 0$, the relationship is trivially satisfied, as d is non-negative, by definition

There are, however, several reasons for using aggregation in this way, related to the goals discussed previously (see Sect. 1).* Reducing extra CPU time has been an implicit goal since the earliest metric indexing structures. Hierarchical decomposition of the search space makes it possible to filter out regions of data at increasing levels of precision, somewhat like what is done in search trees over ordered data.

⟨**note**$_{10}$⟩ Indeed, it may seem like some structures have rather uncritically transplanted the ideas behind search trees to the field of similarity search, even though the advantages of this adaptation are not as obvious as they might seem. In addition to the varying balance between distance computations, CPU time, and I/O, there are such odd phenomena as unbalanced structures being superior to balanced ones in certain circumstances (see the discussion of LC in Sect. 7). ⟨/**note**$_{10}$⟩

*These reasons, or the fact that aggregation in itself can only reduce filtering power, do not seem to be addressed by most authors.

$\hookrightarrow$ When it comes to reducing the number of disk accesses, which is an important consideration for several recent methods, it is also clear that being able to exclude entire disk blocks without examining them (i.e., without reading them into main memory) is vital, and in order to manage this, some form of aggregation is needed.

$\langle$**note**$_{11}\rangle$ Examples of methods that rely on this are the M-tree [78–80] (and its descendants), the MVP-tree [81] and D-index [50–53, 101]. $\langle$/**note**$_{11}\rangle$

$\hookrightarrow$ However, even with a single-minded focus on reducing the number of distance computations, aggregation may be an important factor. Although using regions rather than individual objects reduces the precision of the distance estimates (and, hence, the filtering power), it may also reduce the amount of memory needed for the index structure; if this is the case, methods using aggregation may, given the same amount of memory, actually be able to *improve* upon the filtering power of those without it.

$\langle$**note**$_{12}\rangle$ Chávez et al. [33] discuss this in depth for the case of metric indexing, and the use of compact regions versus pivoting (see Sects. 4 through 6), and show that, in general, if one has an unlimited amount of memory, pivoting will be superior, but that the region-based methods will normally utilize limited memory more efficiently. $\langle$/**note**$_{12}\rangle$

3 The Geometry of Metric Spaces

In order to qualify as a distance function, a function must normally be symmetric and reflexive. If d is a distance defined on the universe $\mathbb{U}$, the pair $(\mathbb{U}, d)$ is called a *distance space*. A *metric space* is a distance space where non-identical objects are separated by positive distances and where there are no "short-cuts": The distance from x to z cannot be improved by going via another point (object) y.* The distance function of a metric space is called (naturally enough) a *metric*. One important property of metric spaces is that subsets (or subspaces) will also be metric, so if a metric is defined over a given data type, a finite data set (and relevant queries) of that data type will form a (finite) metric space, subject to the same constraints.

Metric spaces are a generalization of Euclidean space, keeping some of its well-known geometric properties. These properties allow us to derive certain facts (and from them, upper and lower bounds) without knowing the exact form of the distance in question.

$\langle$**note**$_{13}\rangle$ The theory of metric spaces is extensive (see the tutorial by Semmes [102] for a brief introduction, or the book by Jain and Ahmad [103] for a more

*More precisely, a distance function satisfies $d(x, y) = d(y, x)$ and $d(x, x) = 0$, while a metric also satisfies $d(x, y) > 0$, unless $x = y$, as well as the *triangle inequality*, $d(x, z) \leq d(x, y) + d(y, z)$.

thorough treatment), but the discussion in this chapter focuses on the basic properties of metric spaces, and how they permit the construction of bounds usable for indexing. ⟨/**note**$_{13}$⟩

While the metric properties may not be a perfect fit for modelling our intuition of similarity,

⟨**note**$_{14}$⟩ See, for example, the discussion of Skopal [18] for some background on this issue. One example he gives, illustrating broken triangularity, is that of comparing humans, horses and centaurs. While a human might be quite similar to a centaur, and a centaur looks quite a bit like a horse, most would think a human to be completely different from a horse. ⟨/**note**$_{14}$⟩

↪ nor applicable to all actual similarity retrieval applications,

⟨**note**$_{15}$⟩ One example of a decidedly non-metric distance is the one used in the query-by-whistling system of Arentz et al. [104]. While this distance is non-negative and reflexive, it is neither symmetric nor triangular, and it does not separate non-identical objects. ⟨/**note**$_{15}$⟩

↪ they do capture some notions that seem essential for the intuitive idea of geometric distance. One way to see this is through the metaphor of road networks. The geometry of roads relaxes some of the requirements of Euclidean space, in that the shortest path between two points no longer needs to follow a straight line, but distances still behave "correctly": Unless we have one-way streets, or similar artificial constraints, distance in a road network follows the metric properties exactly. The most interesting of these in this context is, perhaps, triangularity: In finding the distance from A to B we would never follow a more winding road than necessary – we invariably choose the shortest route. Thus, first going from A to C, and subsequently to B, could never give us a smaller sum (that is, a shorter route) than the distance we have defined from A to C. (see Fig. 6). While the distance may not be measured along a straight line, we still measure it along the shortest path possible.

⟨**note**$_{16}$⟩ This is, of course, simply a characterization of metric spaces in terms of (undirected) graphs. It is clear that shortest paths (geodesics) in finite graphs with positive edge weights invariably form metric spaces, but it is interesting to note that the converse is also true: Any finite metric space may be realized as such a graph geodesic [see, e.g., Lemma 3.2.1 of 105, p. 62]. It is also interesting to note that the triangularity of geodesics is preserved in the presence of negative edge weights (that is, shortest paths still cannot be shortened). Similarly, directed triangularity holds for directed graphs. ⟨/**note**$_{16}$⟩

One way of interpreting triangularity is that the concept of overlap becomes meaningful for naturally defined regions such as *metric balls* (everything within a given distance of some center object). If we're allowed to violate triangularity, two seemingly non-overlapping balls could still share objects (see Fig. 7 on the facing page).

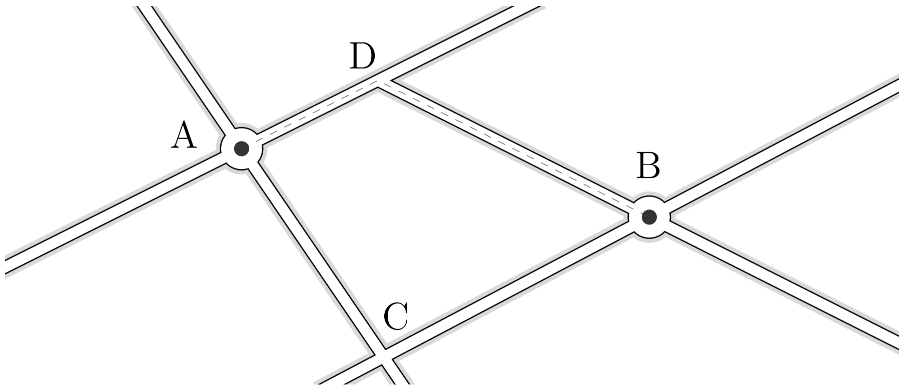


Fig. 6. Triangularity in a road network. The distance from A to B is determined by the shortest route, which goes via D, and it cannot be improved by going via C. More importantly, it cannot be improved by going via D either (because it is already part of the shortest path) – or any other point, for that matter; that is, $d(\mathrm{A}, \mathrm{B}) \leq d(\mathrm{A}, \mathrm{X}) + d(\mathrm{X}, \mathrm{B})$, for any point X in the network

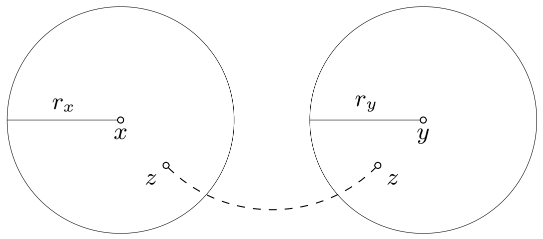


Fig. 7. Illustration of a non-triangular space. Two regions defined as balls (everything within a distance r_x and r_y of x and y, respectively) seemingly don't overlap, as $d(x, y) > r_x + r_y$. Even so, there may be objects (such as z) that are found in both regions

If we assume that our dissimilarity space is, in fact, a metric space, the essential next step is to construct bounds (as discussed previously, in Sect. 2). Two lemmas constructing quite general bounds using only the metric properties can be found in Appendix A. The techniques presented in Sects. 4 through 6 may be seen as special cases of these lemmas, but it should be possible to follow the main text even if the appendix is skipped.

4 Pivoting and Pivot Space

The indexing approaches, and their distance bounds, are generally based on selecting sample objects, also known as *pivots* (or, for the regions discussed in Sects. 5 and 6, sometimes called *centers*). Imagine having selected some pivot object p from your data set, and having pre-computed the distance $d(p, o)$ from the pivot to every other indexed object. This collection of pre-computed distances then becomes your index. When performing a query, you also compute the distance from query to pivot, $d(q, p)$. It is quite easy to see (through some shuffling of the triangle inequality)* that the distance $d(q, o)$ can be lower-bounded as follows (see Fig. 8):

$$\check{d}_p(q, o) = |d(q, p) - d(p, o)| \tag{1}$$

A search may then be performed by scanning through the data set, filtering out objects based on this lower bound. While this does not take into account CPU costs or I/O (see Sect. 1), it does (at least potentially) reduce the total number of distance computations.†

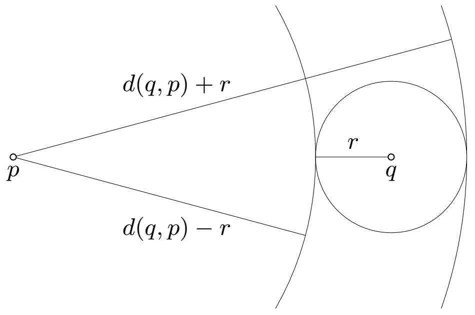


Fig. 8. An illustration of basic pivot filtering, based on (1). For all objects inside the inner ball around p, we have that $d(q, p) - d(p, o) > r$ and for those outside the outer ball, $d(p, o) - d(q, p) > r$. The "maybe" region is the shell between the two, where $|d(q, p) - d(p, o)| \leq r$. Using several lower bounds (that is, pivots) is equivalent to intersecting several shells, more closely approximating the query ball

Perhaps the most obvious way of improving this bound is to combine several bounds – using more than one pivot. For any query, the maximum over all the bounds can then be chosen (as the real distance must, of course, obey all the bounds, and the maximum gives us the closest estimate and therefore the most pruning power). For a set of pivots, P, the lower bound can be defined as follows:

$$\check{d}(q, o) = \max_{p \in P} \check{d}_p(q, o) \tag{2}$$

*$d(q, o) + d(p, o) \geq d(q, p) \Rightarrow d(q, o) \geq d(q, p) - d(p, o)$. Equivalently, $d(q, o) \geq d(p, o) - d(q, p)$, and hence, $d(q, o) \geq |d(q, p) - d(p, o)|$. This is a special case of Lemma 1 in Appendix A.

†An obvious related upper bound, which can be used for inclusion, is $\hat{d}(q, o) = d(q, p) + d(p, o)$, which follows directly from the triangular inequality.

One interesting way of viewing this is as a non-expansive mapping to a vector space, often called *pivot space*. We can define a mapping $f(x) = \langle d(p, x)\rangle_{p \in P}$. Our lower bound then, in fact, becomes simply L_∞.

This, in fact, is the gist of one of the basic metric indexing methods, called LAESA.

⟨**note**$_{17}$⟩ Linear Approximating and Eliminating Search Algorithm [8, 40]. ⟨/**note**$_{17}$⟩

↪ In LAESA, a set of m pivots is chosen from the data set of size n, and an $n \times m$ matrix is filled with the object-to-pivot distances in a preprocessing step. Searching becomes a linear scan through the matrix, filtering out rows based on the lower bound (2). As the focus is entirely on reducing the number of comparisons, the linear scan is merely seen as acceptable extra CPU cycles. Some CPU time can be saved by storing columns separately, sorted by distance from its pivot. In each column, the set of viable objects will be found in a contiguous interval, and this interval can be found by bisection. One version of the structure even maintains pointers from cells in one column to the next, between those belonging to the same data object, permitting more efficient intersection of the candidate sets.

⟨**note**$_{18}$⟩ This is the Spaghettis structure, described by Chávez et al. [89]. ⟨/**note**$_{18}$⟩

↪ The basic idea of LAESA was rediscovered by Filho et al. (who call it the Omni method), who also index the pivot space using B-trees and R-trees, to reduce CPU time and I/O.

⟨**note**$_{19}$⟩ See the paper by Filho et al. [1] for more information. ⟨/**note**$_{19}$⟩

The choice of pivots can have quite an effect on the performance of this basic pivoting scheme. The simplest approach – simply selecting at random – does work, and several heuristics (such as choosing pivots that are far apart or that have similar distances to each other*) have been proposed to improve the filtering power. One approach, in particular, is to heuristically maximize the lower bound directly. Pivots are added to the pivot set one at a time. For each iteration, the choice stands between a set of randomly sampled candidate pivots. In order to evaluate these, a set of *pairs* of objects is sampled randomly from the data, and the average pivot distance (using the tentative pivot set, including the new candidate in question) is computed. The pivot that gets the highest average is chosen, and the next iteration begins.

⟨**note**$_{20}$⟩ For more details on this approach, and a comparison between it and a few similar heuristics, see the paper by Bustos et al. [68]. A recent variation on choosing pivots that are far apart is called Sparse Spatial Selection [69], and it is used in the so-called SSS-Tree [90] and in SSS-LC [91]. ⟨/**note**$_{20}$⟩

The LAESA method is, in fact, based on an older method, called AESA.

*The latter is the approach used by the Omni method.

⟨**note**$_{21}$⟩ Approximating and Eliminating Search Algorithm [41, 42]. ⟨/**note**$_{21}$⟩

$\hookrightarrow$ In AESA, there is no separate pivot set; instead, any data object may be used as a pivot, and the set of pivots used depends on the query object. In order to achieve this flexibility (and the resulting unsurpassed pruning power) one needs to store the distances between all objects in the data set in an $n \times n$ matrix (or, rather, half of that, because of symmetry). In the first iteration, a pivot is chosen arbitrarily, and the data set is filtered. (As we now have the actual distance to the pivot, it can be either included in or excluded from the final result set.) In subsequent iterations, a pivot is chosen among the remaining, unfiltered objects. This object should – in order to maximize pruning power – be as close as possible to the query. This distance is approximated with the pivot distance (2), using the pivot set built so far.

⟨**note**$_{22}$⟩ Actually, a recent algorithm called iAESA [66], managed, as the first method in twenty years, to improve upon AESAs search performance. iAESA uses the same basic strategy as AESA, but uses a different heuristic for selecting pivots, involving the correlation between distance-based permutations of *another* set of pivots. In theory, any cheap and accurate approximation could be used as such a heuristic, potentially improving the total performance. ⟨/**note**$_{22}$⟩

In addition to these main algorithms based on pivoting, and their variations, the basic principles are used as components in more complex, hybrid structures.

⟨**note**$_{23}$⟩ A couple of variations of interest are ROAESA, or Reduced Overhead-AESA [85], and TLAESA, or Tree-LAESA [92, 93], which reduce the CPU time of AESA and LAESA, respectively, to sublinear. Applications of pivoting in hybrid structures include MVP-tree [81, 82], D-index [50–52, 52, 53, 101] and the related, DF-tree [36], PM-tree [84], and CM-tree [47]. ⟨/**note**$_{23}$⟩

5 Metric Balls and Shells

The pivoting scheme can give quite accurate lower bounds, given enough pivots. However, that number can be high – in many cases leading to unrealistically high space requirements.

⟨**note**$_{24}$⟩ Chávez et al. [33] show that the optimal number is logarithmic in the size of the data set ($\Theta(\lg n)$, for n objects). Filho et al. [1], on the other hand, claim that the required number of pivots is proportional to the fractal (which they also call intrinsic) dimensionality of the data set, and that using pivots beyond this is of little use. In other words, this means that the optimal number of pivots is not necessarily related to the size of the data set (that is, it is $\Theta(1)$). See Sect. 7 for a brief discussion of fractal and intrinsic dimensionality. ⟨/**note**$_{24}$⟩

↪ One possible tradeoff is to reduce the number of pivots below the optimal. As noted in Sect. 2, another possibility is to reduce the information available about each object, through aggregation: Instead of storing the exact distance from an object to a pivot, the object is merely placed in a *region*, somehow defined in terms of some of the pivots. This also implies that the relationship between some objects and some pivots can be left completely undefined.

One of the most obvious region choices in a metric space is, perhaps, metric balls. A ball region $[p]_r$ is defined by one pivot p (often referred to as its *center*), along with a so-called *covering radius* r, an upper bound to the distance from the pivot to any object in the region. There are two common ways of partitioning the data using metric ball regions (see Fig. 9). The first, found in the VP-tree (*vantage point tree*), among others, uses a single ball to create two regions: one inside the ball, and one outside it. The other, found in the BS-tree (*bisector tree*), for example, uses a ball for each region. These two basic partitioning schemes are described in more detail in the following.

⟨**note**$_{25}$⟩ For more information about the VP-tree, see the papers by Uhlmann [59] (who calls them simply *metric trees*) and Yianilos [95] (who rediscovered them, and gave them their name). The VP partitioning scheme has been extended in, for example, the Optimistic VP-tree [83] and the MVP-tree [81, 82]. It is extended with a so-called *exclusion zone* in the Excluded Middle Vantage Point Forest [58] (which is in many ways very similar to the more recent D-index, discussed later). The BS-tree was first described by Kalantari and McDonald [45], and the BS partitioning scheme has been adopted by several subsequent structures, including the M-tree [78–80] and its descendants. A recent addition to the BS-tree family is the SSS-Tree [90]. ⟨/**note**$_{25}$⟩

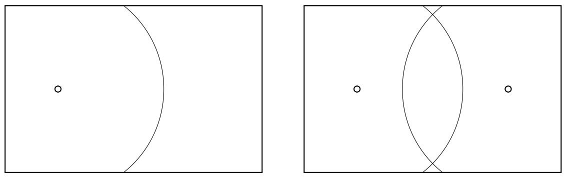

(a) VP-style (b) BS-style

Fig. 9. An illustration of the two basic ball partitioning principles. In VP-style partitioning, the two regions correspond to inside and outside one ball, while in BS-style, each region has its own ball. Multiway versions of both are possible

The VP-tree is a static balanced binary tree structure built as follows, from a given data set: Choose a pivot (or *vantage point*) p and compute the median distance d_m from p to the rest of the data set. Keep p and d_m in the root node, and recursively construct left and right subtrees from the objects that fall inside and outside d_m, respectively.

⟨**note**$_{26}$⟩ A dynamic version of the VP-tree has been proposed by chee Fu et al. [57]. ⟨/**note**$_{26}$⟩

↪ When searching the tree with a query q and a search radius r, the search ball is examined for overlap with the inside and outside regions (there might, of course, be overlap with both), and the overlapping regions are examined recursively. The respective lower bounds for objects in the inside and outside regions are, of course, $d(q,p) - d_m$ and $d_m - d(q,p)$ (see Fig. 10).*

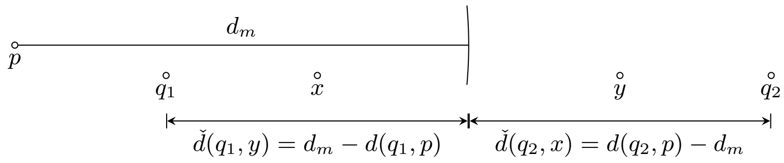

d_m
p
q_1
x
y
q_2
$\check{d}(q_1, y) = d_m - d(q_1, p)$
$\check{d}(q_2, x) = d(q_2, p) - d_m$

Fig. 10. The lower bounds based on VP-style partitioning: The inside region must be checked if $d(q,p) - d_m \leq r$, and the outside region, if $d_m - d(q,p) \leq r$. If q is in the same region as an object, the lower bound becomes negative

While the property of balance might intuitively seem to be an obvious benefit, this may not always be the case. Clearly, the balance does not give us any guaranteed upper bounds on the search time, as we may need to visit multiple (or, indeed, all) subtrees at any point. There may, in fact, be reason to think that unbalanced structures are superior for metric indexing in certain cases. One structure based on this idea is LC (list of clusters).

⟨**note**$_{27}$⟩ Chávez and Navarro [70, 71] give a theoretical analysis of high-dimensional metric spaces (in terms of intrinsic dimensionality, as explained in Sect. 7) as support for their assumption that unbalanced structures may be beneficial. Fredriksson [43] describes several extended versions of the LC structure (HC, AHC and EHC). Marin et al. [91] describe a hybrid of LC and LAESA, using the SSS pivot selection strategy. Another structure that relaxes the balance requirement and achieves improved performance is the DBM-tree [48, 49] ⟨/**note**$_{27}$⟩

↪ Simply put, LC is a highly unbalanced VP-tree, with each left (inside) branch having a fixed size (either in terms of the radius or the number of items). The right (outside) branch contains the rest of the data, and functions almost as a next-pointer in a linked list consisting of the inside region clusters (hence the name). Search in LC is performed in the same manner as in the VP-tree, but it has one property that can give it an edge in some circumstances: If, at any point, the query is completely contained within one of the cluster balls, the entire tail of the list (the outside) can be discarded.

*See also Fig. 5 on page 8, with $f(q) = d(q,p) - d_m$ for the inside region, and $f(q) = d_m - d(q,p)$ for the outside region.

⟨**note**$_{28}$⟩ An interesting point here is that one can use other metric index structures to speed up the building of LC, because one needs to find which objects (of those remaining) are inside the cluster radius, or which k objects are the nearest neighbors of the cluster center – and both of these operations are standard metric queries. ⟨/**note**$_{28}$⟩

The VP-style partitioning can be generalized to multiway trees in at least two ways: Either, one can use multiple radii and partition the data set into shells, or bands, between the radii* (either with a fixed distance between them, or with a fixed number of object in each region), or one can use several pivots in each node. The *multi-vantage point* (MVP) tree uses both of these techniques: Each node in the tree has two pivots (more or less corresponding to two levels of a VP-tree collapsed into a single node) as well as (potentially) several radii for each pivot.

⟨**note**$_{29}$⟩ The MVP-tree is discussed in detail by Bozkaya and Özsoyoglu [81, 82]. In addition to the basic partitioning strategy, each leaf also contains a LAESA structure, filtering with the pivots found on the path from the root to that leaf (thereby reusing distances that have already been computed). ⟨/**note**$_{29}$⟩

The original BS-tree is also a binary tree, but it is dynamic and potentially unbalanced, unlike the VP-tree. The objects in a subtree are contained in a metric ball specific to that tree. The pivot and radius of this ball are kept in the parent node, so each node contains up to two pivots (and corresponding radii). When an object is inserted into a node with zero or one pivots, it is added as a pivot to that node (with an empty subtree). If a node is already full, the pivot is inserted into the subtree of the nearest pivot.

⟨**note**$_{30}$⟩ Note that this decision is actually based on a generalized hyperplane, or bisector – hence the name "bisector tree." Hyperplanes are discussed in more detail in Sect. 6. ⟨/**note**$_{30}$⟩

The BS-tree has some closely related descendants

⟨**note**$_{31}$⟩ Perhaps the most closely related structure is the Voronoi-tree [94, 106], which is essentially a ternary BS-tree with an additional property: When a new leaf is created, the parent pivot (the one closest to the new object) is also added to the leaf. This guarantees that no node can have a greater covering radius than its parent. Another relative is the Monotonous Bisector* Tree (MBS*-tree) [74, 75]. A more recent relative is the BU-tree [46], which is simply a static BS-tree, built bottom-up (hence the name), using clustering techniques. ⟨/**note**$_{31}$⟩

↪ but one of the most well-known structures using BS-style ball partitioning is rarely presented as one of them: The M-tree.

*For more on using shells in indexing, see Fig. 11 on page 19 and the discussion of GNAT in Sect. 6.

⟨**note**$_{32}$⟩ For more details on the M-tree, see the papers by Zezula et al. [80] and Ciaccia et al. [77, 78, 79], for example. Some recent structures based on the M-tree include the QIC-M-tree [25], the Slim-tree [88], the DF-tree [36], the PM-tree [84], the Antipole Tree [44], the M*-tree [76], and the CM-tree [47]. ⟨/**note**$_{32}$⟩

The structures in the M-tree family are, at their core, multiway BS-trees, with each subtree contained in a metric ball, and new objects inserted into the closest ball (if inside, the one with the closest pivot; if outside, the one whose radius will increase the least). While more recents structures based on the M-tree may have extra features (such as special heuristics for insertion or balancing, or AESA-like pivot filtering in each node) this basic structure is common to them all. The main contribution of the M-tree, however, is not simply the use of a multiway tree structure; it is the way it is implemented – as a balanced, disk-based tree. Each node is a disk block, and balancing is achieved through algorithms quite similar to those of the B-tree family (or the spatial index structures of the R-tree family).

⟨**note**$_{33}$⟩ The B-tree and its descendants are described in several basic textbooks on algorithms; a general structure called GiST (Generalized Search Tree, available from `http://gist.cs.berkeley.edu`) implements disk-based B-tree-style balancing for use in index structures in general, and has been used in several published implementations (including the original M-tree). For more information on recent developments in the R-tree family, see the book by Manolopoulos et al. [107]. ⟨/**note**$_{33}$⟩

↪ This means that it is dynamic (with support for insertions and deletions) and that it is usable for large, disk-based data sets.

The iDistance method is a somewhat related approach. Like in the M-tree, the data set is partitioned into ball regions, and one-dimensional pivot filtering is used within each region. However, instead of using a custom data structure, all information about regions and pivot distances is stored in a B$^+$-tree. Each object receives a key that, in effect, consists of two digits: The first is the number of its region, and the second is its distance to the region pivot. A metric search can then be reduced to a set of one-dimensional searches in the B$^+$-tree – one for each region that overlaps the query.

⟨**note**$_{34}$⟩ In practice, the object key is a single value that is constructed by multiplying the region number with a sufficiently large constant, and adding the two. Note that the original description of iDistance focuses on kNN search, but the method works equally well for range queries. For more information on iDistance, including partitioning heuristics, see the paper by Yu et al. [67]. For another way of mapping a metric space onto a B$^+$-tree keys, see the MB$^+$-tree of Ishikawa et al. [73]. ⟨/**note**$_{34}$⟩

A region type closely related to the ball is the *shell*. It is the set difference $[p]_r \setminus [p]_s$ of two metric balls (where $r \geq s$), and a shell region R can be represented with single center, p, and two radii: the *inner* radius, $d^-(p, R)$, and the *outer* radius, $d^+(p,R)$. These will then be lower and upper bounds on the distance $d(p,o)$ for any object o in the region R. Of course, we have similar lower and upper bounds for any object in the query region as well: $d(q,p)-r$ and $d(q,p)+r$. Only if these two distance intervals overlap can there be any hits in R. To put things differently, for each pivot p and corresponding shell region R, we have two lower bounds:

$$\check{d}_1(q,o) = d^-(p,R) - d(q,p)$$
$$\check{d}_2(q,o) = d(q,p) - d^+(p,R)$$

Here, $\check{d}_1$ is non-negative if the query falls inside the shell and $\check{d}_2$ is non-negative if it falls outside* (see Fig. 11).

⟨**note**35⟩ For more on why this is correct, see Lemma 1 on page 28. ⟨/**note**35⟩

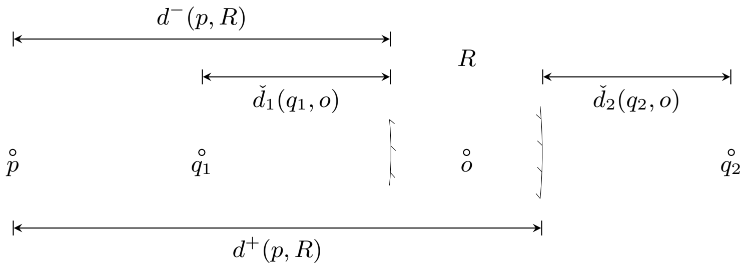


Fig. 11. Lower bounds for a shell region R. As discussed in the text, the shell region gives rise to two bounds: $\check{d}_1(q,o) = d^-(p,R) - d(q,p)$ and $\check{d}_2(q,o) = d(q,p) - d^+(p,R)$. The bound $\check{d}_1$ is only useful if the query falls inside the shell, and $\check{d}_2$ only when it falls outside

For an example of a structure using shell regions, see the discussion of GNAT in the following section.

6 Metric Planes and Dirichlet Domains

A less obvious partitioning choice than metric balls is the generalized hyperplane, also known as the *midset*, between two pivots in a metric space – the set of objects for which the the pivots are equidistant. The two regions generated by this midset consist of the objects closer to one pivot or the other. In the Euclidean plane, this can be visualized by the half-plane on either side of the bisector between two points (see Fig. 12(a) on the following page).

⟨**note**36⟩ Note that in general, the midset itself may be empty. ⟨/**note**36⟩

*If the query falls inside R, both bounds are negative.

$\hookrightarrow$ This space bisection is used by the GH-tree, and the multiway equivalent is used by the structure called the Geometric Near-neighbor Access Tree, or simply GNAT (see Fig. 12(b)).

⟨**note**$_{37}$⟩ For more information on the GH-tree, see the papers by Uhlmann [59, 60]; GNAT is described in detail by Brin [61]. ⟨/**note**$_{37}$⟩

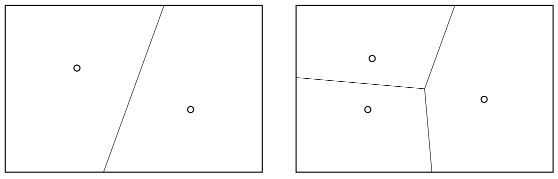


Fig. 12. Illustration of two-way and multiway generalized hyperplane partitioning. In two-way (GH-style) partitioning, two half-spaces are formed, each consisting of objects closer to one of the two pivots. In multiway (GNAT-style) partitioning, this is simply extended to multiple pivots. The resulting regions are called Dirichlet domains

The GH-tree is a close relative of the other basic metric trees – especially the BS-tree. In fact, you might say it's simply a BS-tree without the covering radii. It's a dynamic binary structure where every node has (up to) two pivots and every new object is recursively inserted into the subtree of the closest pivot, just as in the BS-tree. The difference is that no covering radius is maintained or used for searching. Instead, the hyperplane criterion is used directly: If the object o is known to be closer to pivot v than of u, then $(d(q,v) - d(q,u))/2$ is a lower bound on $d(q,o)$, and can be used to filter out the v branch.

⟨**note**$_{38}$⟩ See Lemma 2 on page 28 for an explanation of this bound. Also note that there is no reason why this criterion couldn't be used in *addition* to a covering radius. ⟨/**note**$_{38}$⟩

At core, GNAT is simply a generalization of the GH-style partitioning to multiway trees, partitioning the space into so-called Dirichlet domains (a generalization of Voronoi partitioning). While GNAT is often classified into the "hyperplane family" of metric index methods, this is a bit misleading, as it is almost a closer relative of the BS-tree (and, indeed, the M-tree family) than to the GH-tree, for example. It is built by selecting a set of pivots/centers (heuristically, so they are reasonably distant from each other), allocating objects to their closest pivots, and then processing each region recursively (with a number of pivots proportional to the number of objects in that region, to balance the tree). However, the Dirichlet domains are not used for searching; instead, for each region, a set of covering shells (distance ranges) are constructed – one for each other pivot in that node. If the query ball does not intersect with each of these shells, the given region is discarded. (For a discussion on shell regions, see Sect. 5.)

Recently, Uribe et al. have developed a dynamic version of GNAT called the Evolutionary Geometric Near-neighbor Access Tree, or EGNAT, which is also well-suited for secondary memory and parallelization.

⟨**note**$_{39}$⟩ For more information on EGNAT, see the paper by Uribe et al. [2]. ⟨/**note**$_{39}$⟩

↪ EGNAT is, at heart, simply a GNAT, but it has one addition: All nodes are initially simply buckets (with a given capacity), where the only information stored is the distance to the parent pivot. When a bucket becomes full, it is converted into a GNAT node (with buckets as its children). Searching a GNAT node works just like with GNAT, while searching a bucket uses plain pivoting (using the single pivot available).

Another related structure is the Spatial Approximation (SA) tree.

⟨**note**$_{40}$⟩ For more information on the SA-tree, see the papers by Navarro [86, 87]. Though the originally published structure is static, it has since been extended to admit insertions and deletions [54–56, 64] and combined with pivoting [63, 65]. ⟨/**note**$_{40}$⟩

↪ The SA-tree approximates the metric space similarly to a GNAT node – that is, by a partition into Dirichlet domains – but the tree structure is quite different. Rather than representing a hierarchical decomposition, the tree edges simply connect adjacent regions (with one pivot per node).

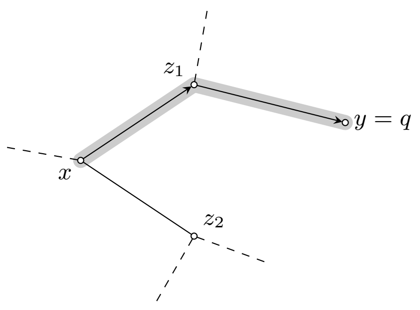
⟨**note**$_{41}$⟩ If you create a graph from the pivots by adding edges between neighboring Dirichlet domains, you get what is often known as the Delaunay graph (or, for the Euclidean plane, the Delaunay triangulation) of the pivot set. The SA-tree will be a spanning tree of this graph. ⟨/**note**$_{41}$⟩

↪ Search is then performed by traversing this tree, moving from region to region.

The SA-tree is built as follows. First a random object is selected as the root, and a set of suitable neighbors are chosen: Every neighbor is required to be closer to the root than to all other neighbors. Also, all *other* objects are closer to at least one of the neighbors than to the root (otherwise they would simply be included as neighbors).

⟨**note**$_{42}$⟩ Note that the neighbor set is defined recursively, and it is not necessarily entirely obvious how to construct it. In fact, there can be several sets satisfying the definition. As finding a minimum set of neighbors it not a trivial task, Navarro [87] uses an incremental heuristic approach: Consider nodes in order of increasing distance from the root, and add them if they are closer to the root than all neighbors added previously. While the method does not guarantee neighbor sets of minimum size, the sets are correct (and usually rather small): Clearly all object that are closer to the root than the other neighbors have been added. Conversely, let's say consider a root r and two neighbors u and v, added in that order. By assumption, $d(u,r) \leq d(v,r)$. We could only have added v if $d(v,r) < d(u,v)$, so both must be closer to r than to each other. ⟨/**note**$_{42}$⟩

↪ Each remaining (non-neighbor) object is assigned to a subset associated with the closest of the neighbors (and subtrees for these subsets are constructed recursively). As a consequence, if we start our traversal at the root, we can always get closer to any object by moving to the neighbor that is closest to our destination (see Fig. 13).



$$\begin{aligned} d(z_1,z_2) &\geq d(z_1,x), d(z_2,x) \\ d(y,x) &\geq d(y,z_1) \\ d(z_1,y) &\leq d(z_2,y) \end{aligned}$$

Fig. 13. For any objects x and y in an SA-tree, y is either a neighbor of x or it is closer to one of x's neighbors than to x. Thus, when the query q is in the data set, we can move gradually closer along the edges. For more realistic searches (involving multiple objects or potential dead ends), backtracking is needed

The basic traversal strategy only works for single objects, and only when searching for objects taken from the data set. If we search for multiple objects (as in a range query) we may need to retrieve objects from multiple branches, and if the query object is unknown, we may hit dead ends that don't end in the desired objects. In either case, we need to add backtracking to our search; or, to put it in the terms used for the other tree-based algorithms: We must traverse the tree, discarding subtrees when possible, using a lower bound on the distance.

The lower bound is twofold: First, a covering radius is kept for each node, and BS-style filtering is used as an initial step (see Sect. 5 for details). Second, the the generalized hyperplane bound (the same as in the GH-tree) is used: $\check{d}(q,o) = (d(q,v) - d(q,u))/2$, for objects closer to (that is, in the subtree of) v.

⟨**note**$_{43}$⟩ See Lemma 2 on page 28 for proof. ⟨/**note**$_{43}$⟩

$\hookrightarrow$ Note that when considering the branch of node v, we could construct a separate bound for each other neighbor, u. Instead, we can get the same filtering power by simply using the u for which $d(q, u)$ is lowest. In fact, at any state of the search, we can choose this u among all the ancestors of the current root, as well as their neighbors, because we know the object o will be closer to v than any of them – and we will already have computed $d(q, u)$ for all of these. The minimum of these distances can be passed along as an extra parameter to the recursive search (updated at each node).

$\langle$**note**$_{44}\rangle$ Although the description of the search algorithm here is formally equivalent to that of Navarro [87], the presentation differs quite a bit. He gives two explanations for the correctness of the search algorithm: one involving a lower bound (similar to the hyperplane argument given here), and one based on the notion of traversing the tree, moving toward a hypothetical object within a distance r of the query. The final result is the same, of course. $\langle/$**note**$_{44}\rangle$

7 Further Approaches and Issues

Although an attempt has been made in this chapter to include the basic principles of all the central approaches to metric indexing, there are, of course, potentially relevant methods that have not been covered, and interesting topics that have not been discussed. As mentioned in the introduction, there are some surveys and textbooks that cover the field much more thoroughly; the remainder of this section elaborates on the the scope of the chapter, and points to some sources of further information.

Other query types.

The range search is, in many ways, the least complex of the distance based query modes. Some other types include the following [see 3].

- Nearest or farthest neighbors: Find the k objects that are nearest to or furthest from the query object.
- Closest or most distant pair: Find the two object that are closest to or most distant from each other. Sometimes known as similarity self-join.
- Ranking and inverse ranking: Traverse the objects in (inverse) order of distance from the query.
- Combinations, such as finding the (at most) k nearest objects within a search radius of r.

Of these, the k-nearest-neighbor (kNN) search (possibly combined with range search) is by far the most common, and perhaps the most intuitive from the user and application point of view. Some principles discussed in this chapter – such as domination and non-expansive mappings – do not transfer directly to the case of kNN search. In order for a transform between distance spaces to preserve the k nearest neighbors, it must preserve the relative distances from the query to these objects; simply underestimating them is not enough to guarantee a correct result. Even so, the techniques behind range search can be used quite effectively to perform kNN searches as well.

Although there are special-purpose kNN mechanisms for some index structures, the general approach is to perform what amounts to a branch-and-bound search: If a given object (or set of objects) cannot improve upon the solution we have found so far, it is discarded. More specifically, we maintain a candidate set as the index is examined, consisting of the (at most) k nearest neighbors found so far, along with a dynamic query radius that tightly covers the current candidates.[*]

As the search progresses, we want to know if we can find any objects that will improve our candidate set; any such objects would have to fall within the current query radius (as they would have to be closer to the query object than the furthest current candidate). Thus, we can use existing radius-based search techniques, shrinking the radius as new candidates are found. The magnitude of the covering radius will then be crucial for search efficiency. It can be kept low for most of the search by heuristically seeking out good candidates early on, for example by visiting regions that are close to the query before those that are further away [3].

Take one of the simplest index structures, the BS-tree (see p. 15). For a range query, subtrees are discarded if their covering balls don't intersect the query. For a kNN search, we use the candidate ball instead: Only if it is intersected is there any hope of finding a closer neighbor.

Another perspective can be found in the original 1-NN versions of pivoting methods such as AESA and its relatives (see p. 13). They alternate between two steps:

1. Heuristically choose a promising neighbor and compute the actual distance to it from the query.
2. Use the pivoting lower bound to eliminate all objects that are further away than the current candidate.

Both the heuristic and the lower bound are based on the set of candidates chosen so far (that is, they are used as pivots). While it might seem like a different approach, this too is really just a matter of maintaining a shrinking query ball with the current candidates, and the method readily generalizes to kNN for $k > 1$.

Other indexing approaches.

While the principles of distance described in Sect. 2 hold in general, in the discussions of particulars in the previous sections, several approaches have deliberately been left out. Some important ones are:

[*]Until we have k candidates, the covering radius is infinite.

- Coordinate-based, or spatial, access methods (see the textbook on the subject by Samet [4] for extensive information on multidimensional indexing). This includes methods in the borderland between purely distance-based methods and spatial ones, such as the M^+- and BM^+-trees of Zhou et al. [5, 6].
- Coordinate-based vector space embeddings, such as FastMap [7]. While pivot space mapping (as used in LAESA [8] and Omni [1]) is an embedding into a vector space ($\mathbb{R}^k$ under L_∞) it is not based on the initial objects having coordinates.
- Approximate methods, such as approximate search with the M-tree [9], MetricMap [10], kNN graphs [see, e.g., 4, pp. 637–641], SASH [11], the proximity preserving order of Chávez et al. [12] or several others [13–15].
- Methods based on stronger assumptions than the metric axioms, such as the growth-restricted metrics of Karger and Ruhl [16].
- Methods based on weaker assumptions than the metric axioms, such as the TriGen method of Skopal [17, 18].*
- Methods exploiting the properties of discrete distances (or discrete metrics, in particular), such as the Burkhart-Keller tree, and The Fixed-Query Tree and its relatives.

  ⟨**note**$_{45}$⟩ These relatives include FHQT, FQA/FMVPA, FHQA, and FQ-MVPT; see the survey by Chávez et al. [33] for details on these discrete-metric methods. ⟨/**note**$_{45}$⟩

- Methods dealing with query types other than range search,† such as the similarity self-join (nearest pair queries) of the eD-index [19], the multi-metric or complex searches of M^3-tree [20] and others [21–24], searching with user-defined metrics, as in the QIC-M-tree [25], or the incremental search of Hjaltason and Samet [26].
- Parallel and distributed methods, using several processors (beyond simple data partitioning), such as GHT* [27], MCAN [28], parallel EGNAT [29] and several other approaches [25, 28, 30–32].

  ⟨**comment**⟩ There are several parallel methods; more will be listed. ⟨/**comment**⟩

Metric space dimension.

It is well known that spatial access methods deal more effectively with low-dimensional vector spaces than high-dimensional ones (one aspect of the so-called *curse of dimensionality*).

*Some approximate methods, such as SASH, kNN graphs, and the proximity preserving order of Chávez et al. also waive the metric axioms in favor of more heuristic ideas of how a distance behaves.

†While kNN is only briefly discussed in this chapter, most of the methods discussed support it.

⟨**note**$_{46}$⟩ For a discussion of how to deal with high-dimensional vector spaces, see the survey by Hetland [100]. It is interesting to note that the intrinsic dimensionality of a vector data set may be lower than its representational dimensionality; that is, the vectors may be mapped faithfully to a lower-dimensional vector space without distorting the distances much. In this case, using a distance-based (metric) index may be quite a bit more efficient than indexing the original dimensions directly with a spatial access method. (See the discussion of intrinsic and fractal dimension in Sect. 7.) ⟨/**note**$_{46}$⟩

↪ Some attempts have been made to generalize the concept of dimension to metric spaces as well, in order to measure how difficult a given space would be to index.

One thing that happens when the dimension increases in Euclidean space, for example, is that all distances become increasingly similar, offset somewhat by any clustering in the data. One hypothesis is that it is this convergence of distances that makes indexing difficult, and this idea is formalized in the *intrinsic dimensionality* of Chávez et al. [33]. Rather than dealing with coordinates, this measure is defined using the distance distribution of a metric (or, in general, distance) space. The intrinsic dimensionality is defined as $\mu^2/2\sigma^2$, where μ and σ^2 are the mean and variance of distribution, respectively. For a set of uniformly random vectors, this is actually proportional to the dimension of the vector space. Note that a large spread of distances results in a lower dimensionality, and vice versa.

An interesting property of distance distributions with a low spread can be seen when considering the cumulative distribution function $N(r)$, the average number of objects returned for a range query with radius r. For r-values around the mean distance, the value of N will increase quite rapidly – the narrower the distribution, the more quickly it will increase. Traina Jr. et al. [34], building on the work on fractal dimensions by Belussi and Faloutsos [35], show that for many synthetic and real-world metric data sets, for suitable ranges of r, the cumulative distribution follows a power-law; that is, $N(r) \in \Theta(r^{\mathcal{D}})$, for some constant $\mathcal{D}$, which they call the *distance exponent*.

⟨**note**$_{47}$⟩ This $\mathcal{D}$ plays a role similar to that of the 'correlation' fractal dimension of a vector space, also known as D_2 [35]. The fractal dimension of a metric space may be found in quadratic time [see, e.g., 108], but the method of Traina Jr. et al. [34] is based on a linear time approximation. Filho et al. [1] use the concept of fractal space in their analysis of the required number of pivots (or foci) for optimum performance of the Omni method (see note 24). ⟨/**note**$_{47}$⟩

↪ They show how to estimate the distance exponent, and demonstrate that it is closely related to, among other things, the number of disk accesses needed when performing range queries using the M-tree.

⟨**comment**⟩ The following link between emptiness and ball overlap may be tenuous. It might be more instructive to link the exponential increase in ball surface to overlap. ⟨/**comment**⟩

Another, rather different, hypothesis is that the problem with high dimensionality in vector spaces is the abundance of emptiness: As the dimension grows, the ratio of data to the sheer volume of space falls exponentially. It becomes hard to create tight regions around subsets of the data, and region overlaps abound. The *ball-overlap factor* (BOF) of Skopal [18] tackles this problem head-on, and measures the relative frequency of overlap between suitably chosen ball regions. BOF_k is defined as the relative frequency of overlap between balls that each cover an object and its k nearest neighbors. In other words, it predicts the likelihood that two rather arbitrary ball-shaped regions will overlap. This factor can describe both the degree of overlap by distinct regions in an index structure, and the the probability that a query will have to investigate irrelevant ball regions (only non-overlapping regions can be discarded).

Method quality.

In the introduction, the main goals and quality measures used in the development of metric indexing methods were briefly discussed, but the full picture is rather complex. The theoretical analysis of these structures can be quite difficult, so experimental validation becomes essential. There are some theoretically defined measures such as the fat and bloat factors and prunability of Traina Jr. et al. [36, 37], but even such properties are generally established empirically for a given structure.

⟨$\mathbf{note}_{48}$⟩ Note that Chávez et al. [33] give some general bounds for pivoting and region-based methods based on intrinsic dimensionality, but these are not really fine-grained enough to gives us definite comparisons of methods. ⟨/$\mathbf{note}_{48}$⟩

↪ Moret [38] gives some reasons why asymptotic analysis alone may not be enough for algorithm studies in general (the worst-case behavior may be restricted to a very small subset of instances and thus not be at all characteristic of instances encountered in practice, and even in the absence of these problems, deriving tight asymptotic bounds may be very difficult). Given a sufficiently problematic metric space, a full linear scan can never be avoided, so the non-asymptotic (primarily empirical) analysis may be particularly relevant for metric indexing.

Even beyond the specifics of a given data set (including measures such as intrinsic dimensionality, fractal dimension, and ball-overlap factor) there are, of course, the real-world issues that plague all theoretical studies of algorithms, such as caching and the memory hierarchy – issues that are not easily addressed in terms of basic principles (at least not given the current state of the theory of metric index structures) and therefore have been omitted from this chapter.

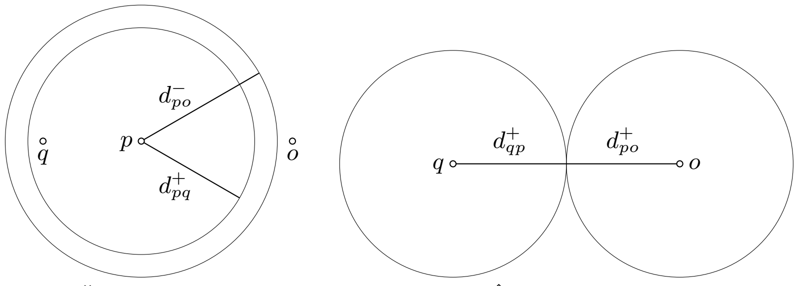
A Bounding Lemmas, with Proofs

The following two lemmas give us some useful bounds for use in metric indexing. For a more detailed discussions, see, for example, the survey by Hjaltason and Samet [3] or the textbook by Zezula et al. [39]. Lemmas 1 and 2 are illustrated in figures 14 and 15, respectively.

Lemma 1 (Ball lemma). *Let o, p and q be objects in $\mathbb{U}$, and let d be a metric over $\mathbb{U}$. For any objects u, v in $\mathbb{U}$, assume that the the value of $d(u,v)$ is known to be in the range $[d^-_{uv}, d^+_{uv}]$. The value of $d(q,o)$ may then be bounded as follows:*

$$\max\{0, d^-_{po} - d^+_{pq}, d^-_{qp} - d^+_{op}\} \leq d(q,o) \leq d^+_{qp} + d^+_{po}\,.$$

Proof. From the triangle inequality we have $d(p,o) \leq d(p,q) + d(q,o)$, which gives us $d(q,o) \geq d(p,o) - d(p,q) \geq d^-_{po} - d^+_{pq}$. Similarly, we have $d(q,p) \leq d(q,o)+d(o,p)$, which gives us $d(q,o) \geq d(q,p)-d(q,p) \geq d^-_{qp}-d^+_{op}$. Finally, the upper bound follows directly from the triangle inequality: $d(q,o) \leq d(q,p) + d(p,o) \leq d^+_{qp} + d^+_{po}$.



(a) $\check{d}(q,o) = d^-_{po} - d^+_{pq}$ (b) $\hat{d}(q,o) = d^+_{qp} + d^+_{po}$

Fig. 14. Illustration of Lemma 1. (a) The bounds on $d(o,p)$ and $d(q,p)$ define two shell regions around p: one containing o and one containing q, and the underestimates are given by the distances from the outer one to the inner one. (b) The upper bound is also given by two shell regions: one around o and one around q, both containing p; because these must overlap in at least one point (that is, p), the overestimate is simply the sum of their (outer) radii

Lemma 2 (Plane lemma). *Let o, q, u and v be objects in $\mathbb{U}$ and let $d(o,v) \leq d(u,o)$. We can then bound $d(q,o)$ as follows:*

$$\max\{(d(q,v) - d(q,u))/2, 0\} \leq d(q,o)\,.$$

Proof. From the triangle inequality, we have $d(q, v) \leq d(q, o) + d(o, v)$, which yields $d(q, v) - d(q, o) \leq d(o, v)$. When combined with $d(u, o) \leq d(q, u) + d(q, o)$ (from the triangle inequality) and $d(o, v) \leq d(u, o)$, we obtain $d(q, v) - d(q, o) \leq d(q, u) + d(q, o)$. Rearranging yields $d(q, v) - d(q, u) \leq 2d(q, o)$, which yields the first combined of the lower bound, the second component being furnished by non-negativity.

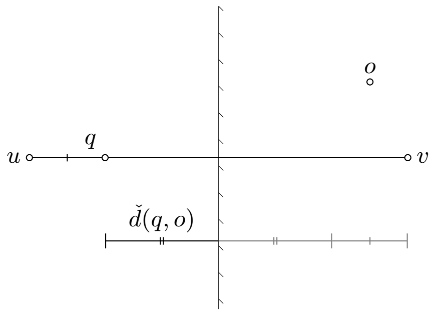


Fig. 15. Illustration of Lemma 2 ($\check{d}(q, o) = (d(q, v) - d(q, u)/2)$). The lower bound may intuitively be seen as a bound on the distance from q (on the u-side) to the "plane" of points midway from u to v (note that no such points may actually exist). As the object o is on the v-side, this is a lower bound on $d(q, o)$. The figure shows the case where q is on the metric interval between u and v (that is, $d(u, v) = d(u, q) + d(q, v)$); other placements of q for the same bound (the hyperbola in the figure) would be further away from the v-region

B An Overview of the Indexing Methods

Many methods are discussed in this chapter, either in the main text or in the end notes. Some important ones are summarized in Table 1 on the next page, with a brief indication of their structure and functionality. The structural properties indicate the use of pivoting (P), BS-style ball partitioning (BS), VP-style ball partitioning (VP), generalized hyperplane partitioning (GH), and multiway partitioning (MW). The functional features indicate whether the method is dynamic (D), whether it is designed to reduce extra CPU cost (CPU) or memory use and/or disk accesses (I/O). A bullet (•) indicates the presence of a property, while a circle (◦) indicates partial (or implicit) presence. This table, of course, only gives the coarsest of overviews of the differences between the methods, certainly does not cover all their unique features, and is not meant as a comparison of their relative merits.

Table 1. Some structural properties and functional features of some of the metric indexing methods discussed in this chapter. The numbers refer to the bibliography

		Structure					**Functionality**		
Method		P	BS	VP	GH	MW	D	CPU	I/O
AESA	40–42	●							
AHC	43	●		●				●	
Antipole Tree	44	●	●		○	○		●	
BS-Tree	45		●		○		●	●	
BU-Tree	46		●		○			●	
CM-Tree	47	●	●		○	●	●	●	●
DBM-Tree	48, 49	○	●		○	●	●	●	●
DF-Tree	36	●	●		○	●	●	●	●
D-Index	50–53	●		●		●	○	●	●
DSA-Tree	54–56				●	●	●	●	○
DSAP-Tree	63–65	●			●	●	●	●	○
DVP-Tree	57			●		●	●	●	●
EGNAT	2	○		●	○	●	●	●	●
EHC	43			●		○		●	
EM-VP-Forest	58			●				●	
GH-Tree	59, 60				●		●	●	
GNAT	61			●	○	●		●	
HC	43, 62			●				●	
iAESA	66	●							
iDistance	67	○	●		○	●	●	●	●
LAESA	8, 40, 68, 69	●					●		○
LC	70, 71			●				●	○
Linear Scan	72						●		
MB$^+$-Tree	73			●		●	●	●	●
MB*-Tree	74, 75		●		○			●	
M*-Tree	76	●	●		○	●	●	●	●
M-Tree	77–80	○	●		○	●	●	●	●
MVP-Tree	81, 82	●		●		●	●	●	●
Omni	1	●					●	●	●
Opt-VP-Tree	83			●		●	●	●	●
PM*-Tree	76	●	●		○	●	●	●	●
PM-Tree	84	●	●		○	●	●	●	●
ROAESA	85	●						●	
SA-Tree	86, 87				●	●		●	
Slim-Tree	37, 88	○	●		○	●	●	●	●
Spaghettis	89	●					●	●	
SSS-Tree	90		●		●	●		●	
SSS-LC	91	●		●			○	●	●
TLAESA	92, 93	●					●	●	
Voronoi-Tree	94		●		○	○	●	●	
VP-Tree	59, 60, 95			●				●	

Acknowledgements

The author would like to thank Simon Jonassen for useful and constructive comments.

Notes

⟨**comment**⟩ This is where the end notes go in the final version. ⟨/**comment**⟩

References

[1] Roberto Figueira Santos Filho, Agma Traina, Caetano Traina Jr., and Christos Faloutsos. Similarity search without tears : the OMNI-family of all-purpose access methods. In *Proceedings of the 17th International Conference on Data Engineering, ICDE*, pages 623–630, 2001.

[2] Roberto Uribe, Gonzalo Navarro, Ricardo J. Barrientos, and Mauricio Marín. An index data structure for searching in metric space databases. In Vassil N. Alexandrov, Geert Dick van Albada, Peter M. A. Sloot, and Jack Dongarra, editors, *Proceedings of the 6th International Conference of Computational Science, ICCS*, volume 3991 of *Lecture Notes in Computer Science*, pages 611–617. Springer, 2006.

[3] Gisli R. Hjaltason and Hanan Samet. Index-driven similarity search in metric spaces. *ACM Transactions on Database Systems, TODS*, 28(4): 517–580, December 2003.

[4] Hanan Samet. *Foundations of Multidimensional and Metric Data Structures*. Morgan Kaufmann, 2006.

[5] Xiangmin Zhou, Gouren Wang, Jeffrey Xu Yu, and Ge Yu. M$^+$-tree: A new dynamical multidimensional index for metric spaces. In Xiaofang Zhou and Klaus-Dieter Schewe, editors, *Proceedings of the 14th Australasian Database Conference, ADC*, volume 17 of *Conferences in Research and Practice in Information Technology*, 2003.

[6] Xiangmin Zhou, Guoren Wang, Xiaofang Zhou, and Ge Yu. BM+-tree: A hyperplane-based index method for high-dimensional metric spaces. In Lizhu Zhou, Beng Chin Ooi, and Xiaofeng Meng, editors, *Proceedings of the 10th International Conference on Database Systems for Advanced Applications, DASFAA*, volume 3453 of *Lecture Notes in Computer Science*, page 398. Springer, 2005.

[7] Christos Faloutsos and King-Ip Lin. FastMap: A fast algorithm for indexing, data-mining and visualization of traditional and multimedia. In Michael J. Carey and Donovan A. Schneider, editors, *Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data*, pages 163–174, San Jose, California, 1995.

[8] María Luisa Micó and José Oncina. A new version of the nearest-neighbour approximating and eliminating search algorithm (AESA) with linear preprocessing time and memory requirements. *Pattern Recognition Letters*, 15(1):9–17, January 1994.

[9] P. Zezula, P. Savino, G. Amato, and F. Rabitti. Approximate similarity retrieval with M-Trees. *The VLDB Journal*, 7(4):275–293, 1998.

[10] J. T. L. Wang, Xiong Wang, D. Shasha, and Kaizhong Zhang. Metricmap: an embedding technique for processing distance-based queries in metric spaces. *IEEE Transactions on Systems, Man and Cybernetics, Part B*, 35(5):973–987, October 2005.

[11] M. E. Houle and J. Sakuma. Fast approximate similarity search in extremely high-dimensional data sets. In *Proceedings of the 21st IEEE International Conference on Data Engineering, ICDE*, 2005.

[12] Edgar Chávez, Karina Figueroa, and Gonzalo Navarro. Proximity searching in high dimensional spaces with a proximity preserving order. In *Proceedings of the 4th Mexican International Conference on Artificial Intelligence, MICAI*, volume 3789 of *Lecture Notes in Computer Science*, pages 405–414. Springer, 2005.

[13] G. Amato. *Approximate similarity search in metric spaces*. PhD thesis, Computer Science Department – University of Dortmund, August-Schmidt-Str. 12, 44221, Dortmund, Germany, 2002.

[14] G. Amato, F. Rabitti, P. Savino, and P. Zezula. Region proximity in metric spaces and its use for approximate similarity search. *ACM Transactions on Information Systems, TOIS*, 21(2):192–227, 2003.

[15] P. Ciaccia and M. Patella. PAC nearest neighbor queries: Approximate and controlled search in high-dimensional and metric spaces. In IEEE Computer Society, editor, *Proceedings of the 16th International Conference on Data Engineering, ICDE*, pages 244–255, 2000.

[16] David R. Karger and Matthias Ruhl. Finding nearest neighbors in growth-restricted metrics. In *Proceedings of the 34th annual ACM symposium on Theory of computing*, pages 741–750. ACM Press, 2002.

[17] Tomáš Skopal. On fast non-metric similarity search by metric access methods. In Yannis Ioannidis, Marc H. Scholl, Joachim W. Schmidt, Florian Matthe, Mike Hatzopoulos, Klemens Boehm, Alfons Kemper, Torsten Grust, and Christian Boehm, editors, *Proceedings of the 10th International Conference on Extending Database Technology*, volume 3896 of *Lecture Notes In Computer Science*, pages 718–736. Springer-Verlag, March 2006.

[18] Tomáš Skopal. Unified framework for exact and approximate search in dissimilarity spaces. *ACM Transactions on Database Systems, TODS*, 32(4), 2007.

[19] Vlastislav Dohnal, Claudio Gennaro, and Pavel Zezula. Similarity join in metric spaces using eD-index. In *Proceedings of the 14th Indernational Conference on Database and Expert Systems Applications,*

DEXA, volume 2736 of *Lecture Notes in Computer Science*, pages 484–493. Springer, September 2003.

[20] Benjamin Bustos and Tomáš Skopal. Dynamic similarity search in multi-metric spaces. In *Proceedings of the 8th ACM international workshop on Multimedia information retrieval*. ACM Press, 2006.

[21] P. Ciaccia, D. Montesi, W. Penzo, and A. Trombetta. Imprecision and user preferences in multimedia queries: A generic algebraic approach. In *Proceedings of the 1st International Symposium on Foundations of Information and Knowledge Systems, FoIKS*, number 1762 in Lecture Notes in Computer Science, pages 50–71. Springer, 2000.

[22] P. Ciaccia, M. Patella, and P. Zezula. Processing complex similarity queries with distance-based access methods. In *Proceedings of the 6th International Conference on Extending Database Technology, EDBT*, number 1377 in Lecture Notes in Computer Science, pages 9–23. Springer, 1998.

[23] R Fagin. Combining fuzzy nformation from multiple systems. In *Proceedings of the 15th ACM Symposium on Principles of Database Systems, PODS*, pages 216–226. ACM Press, 1996.

[24] R Fagin. Fuzzy queries in multimedia database systems. In *Proceedings of the 17th ACM Symposium on Principles of Database Systems, PODS*, pages 1–10. ACM Press, 1998.

[25] Paolo Ciaccia and Marco Patella. Searching in metric spaces with user-defined and approximate distances. *ACM Transactions on Database Systems (TODS)*, 27(4):398–437, 2002.

[26] G. R. Hjaltason and H. Samet. Incremental similarity search in multimedia databases. Technical Report CS-TR-4199, Computer Science Department, University of Maryland, College Park, 2000.

[27] Michal Batko, Claudio Gennaro, Pasquale Savino, and Pavel Zezula. Scalable similarity search in metric spaces. In *Proceedings of the DELOS Workshop on Digital Library Architectures*, pages 213–224, 2004.

[28] Fabrizio Falchi, Claudio Gennaro, and Pavel Zezula. Nearest neighbor search in metric spaces through content-addressable networks. *Information Processing & Management*, 43(3):665–683, 2007.

[29] Mauricio Marín, Roberto Uribe, and Ricardo J. Barrientos. Searching and updating metric space databases using the parallel EGNAT. In *Proceedings of the International Conference on Computational Science*, number 4487 in Lecture Notes in Computer Science, pages 229–236. Springer, 2007.

[30] A. Alpkocak, T. Danisman, and T. Ulker. A parallel similarity search in high dimensional metric spaces using M-Tree. In *Proceedings of the NATO Advanced Research Workshop on Advanced Environments, Tools, and Applications for Cluster Computing – Revised Papers, IWCC*, number 2326 in Lecture Notes in Computer Science. Springer, 2002.

[31] David Novak and Pavel Zezula. M-Chord: a scalable distributed similarity search structure. In *Proceedings of the 1st International Conference on Scalable Information Systems*, 2006.

[32] P. Zezula, P. Savino, F. Rabitti, G. Amato, and P. Ciaccia. Processing M-Trees with parallel resources. In *Proceedings of the 8th International Workshop on Research Issues in Data Engineering, RIDE*, pages 147–154. IEEE Computer Society, 1998.

[33] Edgar Chávez, Gonzalo Navarro, Ricardo Baeza-Yates, and José Luis Marroquín. Searching in metric spaces. *ACM Computing Surveys*, 33 (3):273–321, September 2001.

[34] Caetano Traina Jr., J. M. Traina, and Christos Faloutsos. Distance exponent: a new concept for selectivity estimation in metric trees. In *Proceedings of the International Conference on Data Engineering, ICDE*, 2000.

[35] A. Belussi and C. Faloutsos. Estimating the selectivity of spatial queries using the correlation fractal dimension. In *Proceedings of the International Conference on Very Large Databases, VLDB*, 1995.

[36] Caetano Traina Jr., Agma Traina, Roberto Figueira Santos Filho, and Christos Faloutsos. How to improve the pruning ability of dynamic metric access methods. In *Proceedings of the eleventh international conference on Information and knowledge management*, pages 219–226, 2002.

[37] Caetano Traina Jr., Agma J. M. Traina, Bernhard Seeger, and Christos Faloutsos. Slim-trees: High performance metric trees minimizing overlap between nodes. In *Proceedings of the 7th International Conference on Extending Database Technology, EDBT*, volume 1777 of *Lecture Notes in Computer Science*, pages 51–65. Springer-Verlag, 2000.

[38] Bernard M. E. Moret. Towards a discipline of experimental algorithmics. In *Data Structures, Near Neighbor Searches, and Methodology: F ifth and Sixth DIMACS Implementation Challenges*, volume 59, pages 197–214. Americal Mathematical Society, 2002.

[39] Pavel Zezula, Giuseppe Amato, Vlastislav Dohnal, and Michal Batko. *Similarity Search : The Metric Space Approach*. Springer, 2006.

[40] Juan Ramón Rico-Juan and Luisa Micó. Comparison of AESA and LAESA search algorithms using string and tree-edit-distances. *Pattern Recognition Letters*, 24(9–10):1417–1426, 2002.

[41] Enrique Vidal. New formulation and improvements of the nearest-neighbour approximating and eliminating search algorithm (AESA). *Pattern Recognition Letters*, 15(1):1–7, January 1994.

[42] Enrique Vidal Ruiz. An algorithm for finding nearest neighbours in (approximately) constant average time. *Pattern Recognition Letters*, 4 (3):145–157, 1986.

[43] Kimmo Fredriksson. Engineering efficient metric indexes. *Pattern Recognition Letters*, 2006.

[44] Domenico Cantone, Alfredo Ferro, Alfredo Pulvirenti, Diego Reforgiato Recupero, and Dennis Shasha. Antipole tree indexing to support range search and k-nearest neighbor search in metric spaces. *IEEE Transactions on Knowledge and Data Engineering*, 17(4):535–550, April 2005.

[45] Iraj Kalantari and Gerard McDonald. A data structure and an algorithm for the nearest point problem. *IEEE Transactions on Software Engineering*, 9(5):631–634, 1983.

[46] Bing Liu, Zhihui Wang, Xiaoming Yang, Wei Wang, and Baile Shi. A bottom-up distance-based index tree for metric space. In *Rough Sets and Knowledge Technology*, volume 4062 of *Lecture Notes in Computer Science*, pages 442–449. Springer, 2006.

[47] Lior Aronovich and Israel Spiegler. Efficient similarity search in metric databases using the CM-tree. *Data & Knowledge Engineering*, 2007.

[48] Marcos R. Vieira, Caetano Traina Jr., Fabio Jun Takada Chino, and Agma J. M. Traina. DBM-tree : a dynamic metric access method sensitive to local density data. In *Proceedings of the 19th Brazilian Symposium on Databases (SBBD).* UnB, 2004.

[49] Marcos R. Vieira, Caetano Traina Jr., Fabio Jun Takada Chino, and Agma Juci Machado Traina. DBM-tree : trading height-balancing for performance in metric access methods. *Journal of the Brazilian Computer Society*, 11(3):20–39, 2006.

[50] Vlastislav Dohnal. An access structure for similarity search in metric spaces. In Wolfgang Lindner, Marco Mesiti, Can Türker, Yannis Tzitzikas, and Athena Vakali, editors, *EDBT Workshops*, volume 3268 of *Lecture Notes In Computer Science*, pages 133–143. Springer, 2004.

[51] Vlastislav Dohnal. *Indexing Structures for Searching in Metric Spaces.* PhD thesis, Masaryk University, Faculty of Informatics, 2004.

[52] Vlastislav Dohnal, Claudio Gennaro, Pasquale Savino, and Pavel Zezula. Separable splits of metric data sets. In *Proceedings of the Nono Convegno Nazionale Sistemi Evoluti per Basi di Dati, SEBD*, June 2001.

[53] Vlastislav Dohnal, Claudio Gennaro, Pasquale Savino, and Pavel Zezula. D-index: Distance searching index for metric data sets. *Multimedia Tools and Applications*, 21(1):9–33, 2003.

[54] Gonzalo Navarro and Nora Reyes. Dynamic spatial approximation trees. In *Proceedings of the XXI Conference of the Chilean Computer Science Society, SCCC*, pages 213–222, 2001.

[55] Gonzalo Navarro and Nora Reyes. Fully dynamic spatial approximation trees. In *Proceedings of the 9th International Symposium on String Processing and Information Retrieval, SPIRE*, volume 2476 of *Lecture Notes in Computer Science*, pages 254–270, 2002.

[56] Gonzalo Navarro and Nora Reyes. Improved deletions in dynamic spatial approximation trees. In *Proceedings of the XXIII International Conference of the Chilean Computer Science Society, SCCC*, 2003.

[57] Ada Wai chee Fu, Polly Mei shuen Chan, Yin-Link Cheung, and Yiu Sang Moon. Dynamic vp-tree indexing for n-nearest neighbor search given pair-wise distances. *The VLDB Journal*, 9(2):154–173, July 2000.

[58] Peter N. Yianilos. Excluded middle vantage point forests for nearest neighbor search. Technical report, NEC Research Institute, 1999.

[59] Jeffrey K. Uhlmann. Metric trees. *Applied Mathematics Letters*, 4(5): 61–62, 1991.

[60] Jeffrey K. Uhlmann. Satisfying general proximity/similarity queries with metric trees. *Information Processing Letters*, 40(4):175–179, November 1991.

[61] Sergey Brin. Near neighbor search in large metric spaces. In Umeshwar Dayal, Peter M. D. Gray, and Shojiro Nishio, editors, *Proceedings of 21th International Conference on Very Large Data Bases, VLDB*, pages 574–584. Morgan Kaufmann, 1995.

[62] Kimmo Fredriksson. Exploiting distance coherence to speed up range queries in metric indexes. *Information Processing Letters*, 95(1):287–292, 2005.

[63] Diego Arroyuelo, Francisca Muñoz, Gonzalo Navarro, and Nora Reyes. Memory-adaptative dynamic spatial approximation trees. In *Proceedings of the 10th International Symposium on String Processing and Information Retrieval, SPIRE*, 2003.

[64] Diego Arroyuelo, Gonzalo Navarro, and Nora Reyes. Fully dynamic and memory-adaptative spatial approximation trees. In *Proceedings of the Argentinian Congress in Computer Science, CACIC*, pages 1503–1513, 2003.

[65] Edgar Chavez, Norma Herrera, and Nora Reyes. Spatial approximation + sequential scan = efficient metric indexing. In *Proceedings of the XXIV International Conference of the Chilean Computer Science Society, SCCC*, 2004.

[66] Karina Figueroa, Edgar Chávez, Gonzalo Navarro, and Rodrigo Paredes. On the least cost for proximity searching in metric spaces. In Carme Àlvarez and Maria Serna, editors, *Proceedings of the 5th International Workshop on Experimental Algorithms*, volume 4007 of *Lecture Notes in Computer Science*, pages 279–290. Springer, 2006.

[67] Cui Yu, Beng Chin Ooi, Kian-Lee Tan, and H. V. Jagadish. Indexing the distance: An efficient method to KNN processing. In *Proceedings of the 27th International Conference on Very Large Data Bases, VLDB*, pages 421–430, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc.

[68] Benjamin Bustos, Gonzalo Navarro, and Edgar Chávez. Pivot selection techniques for proximity searching in metric spaces. *Pattern Recognition Letters*, 24(14):2357–2366, October 2003.

[69] Oscar Pedreira and Nieves R. Brisaboa. Spatial selection of sparse pivots for similarity search in metric spaces. In *Proceedings of the 33rd Conference on Current Trends in Theory and Practice of Computer Science*,

number 4362 in Lecture Notes in Computer Science, pages 434–445, 2007.

[70] Edgar Chávez and Gonzalo Navarro. An effective clustering algorithm to index high dimensional metric spaces. In *Proceedings of the 6th International Symposium on String Processing and Information Retrieval, SPIRE*, pages 75–86. IEEE Computer Society, 2000.

[71] Edgar Chávez and Gonzalo Navarro. A compact space decomposition for effective metric indexing. *Pattern Recognition Letters*, 26(9):1363–1376, July 2005.

[72] Kevin Beyer, Jonathan Goldstein, Raghu Ramakrishnan, and Uri Shaft. When is "nearest neighbor" meaningful? In *Proceedings of the 7th International Conference on Database Theory*, volume 1540 of *Lecture Notes In Computer Science*, pages 217–235, London, UK, 1999. Springer-Verlag.

[73] Masahiro Ishikawa, Hanxiong Chen, Kazutaka Furuse, Jeffrey Xu Yu, and Nobuo Ohbo. MB$^+$-tree: A dynamically updatable metric index for similarity search. In H. Lu and A. Zhou, editors, *Proceedings of the First International Conference on Web-Age Information Management*, volume 1846 of *Lecture Notes in Computer Science*, page 356. Springer, 2000.

[74] Hartmut Noltemeier, Knut Verbarg, and Christian Zirkelbach. A data structure for representing and efficient querying large scenes of geometric objects: MB* trees. In Gerald E. Farin, Hans Hagen, Hartmut Noltemeier, and Walter Knödel, editors, *Geometric Modelling*, volume 8 of *Computing Supplement*. Springer, 1992.

[75] Hartmut Noltemeier, Knut Verbarg, and Christian Zirkelbach. Monotonous bisector* trees : a tool for efficient partitioning of complex scenes of geometric objects. In Burkhard Monien and Thomas Ottmann, editors, *Data Structures and Efficient Algorithms*, volume 594 of *Lecture Notes In Computer Science*, pages 186–203. Springer, 1992.

[76] T. Skopal and D. Hoksza. Improving the performance of M-tree family by nearest-neighbor graphs. In *Proceedings of the Eleventh East-European Conference on Advances in Databases and Information Systems, ADBIS*, 2007.

[77] Paolo Ciaccia, Marco Patella, Fausto Rabitti, and Pavel Zezula. Indexing metric spaces with M-tree. In Matteo Cristiani and Letizia Tanca, editors, *Atti del Quinto Convegno Nazionale su Sistemi Evoluti per Basi di Dati (SEBD'97)*, pages 67–86, Verona, Italy, June 1997.

[78] Paolo Ciaccia, Marco Patella, and Pavel Zezula. M-tree: An efficient access method for similarity search in metric spaces. In *Proceedings of the 23rd International Conference on Very Large Data Bases, VLDB*, pages 426–435. Morgan Kaufmann, 1997.

[79] Paolo Ciaccia, Marco Patella, and Pavel Zezula. A cost model for similarity queries in metric spaces. In *Proceedings of the 17th Symposium on Principles of Database Systems*, pages 59–68, 1998.

[80] Pavel Zezula, Paolo Ciaccia, and Fausto Rabitti. M-tree: A dynamic index for similarity queries in multimedia databases. Technical Report 7, HERMES ESPRIT LTR Project, 1996.

[81] Tolga Bozkaya and Meral Özsoyoglu. Distance-based indexing for high-dimensional metric spaces. In *Proceedings of the 1997 ACM SIGMOD international conference on Management of data*, pages 357–368, New York, NY, USA, 1997. ACM Press.

[82] Tolga Bozkaya and Meral Özsoyoglu. Indexing large metric spaces for similarity search queries. *ACM Transactions on Database Systems, TODS*, 24(3), 1999.

[83] Tzi-cker Chiueh. Content-based image indexing. In *Proceedings of the Twentieth International Conference on Very Large Databases, VLDB*, pages 582–593, Santiago, Chile, 1994.

[84] Tomáš Skopal. Pivoting M-tree: A metric access method for efficient similarity search. In V. Snášel, J. Pokorn'y, and K. Richta, editors, *Proceedings of the Annual International Workshop on DAtabases, TExts, Specifications and Objects (DATESO)*, volume 98 of *CEUR Workshop Proceedings*, Desna, Czech Republic, April 2004. Technical University of Aachen (RWTH).

[85] J. M. Vilar. Reducing the overhead of the AESA metric-space nearest neighbour searching algorithm. *Information Processing Letters*, 56(5): 265–271, December 1995.

[86] Gonzalo Navarro. Searching in metric spaces by spatial approximation. In *Proceedings of the 6th International Symposium on String Processing and Information Retrieval, SPIRE*, pages 141–148. IEEE Computer Society, 1999.

[87] Gonzalo Navarro. Searching in metric spaces by spatial approximation. *The VLDB Journal*, 11(1):28–46, August 2002.

[88] Tomáš Skopal, Jaroslav Pokorný, Michal Krátký, and Václav Snášel. Revisiting M-tree building principles. In *Advances in Databases and Information Systems*, volume 2798 of *Lecture Notes In Computer Science*, pages 148–162, September 2003.

[89] Edgar Chávez, José Luis Marroquín, and Ricardo Baeza-Yates. Spaghettis: An array based algorithm for similarity queries in metric spaces. In *Proceedings of the String Processing and Information Retrieval Symposium & International Workshop on Groupware (SPIRE)*, pages 38–46. IEEE Computer Society, September 1999.

[90] Nieves Brisaboa, Oscar Pedreira, Diego Seco, Roberto Solar, and Roberto Uribe. Clustering-based similarity search in metric spaces with sparse spatial centers. In *Proceedings of the 34th Conference on Current Trends in Theory and Practice of Computer Science*, number 4910 in Lecture Notes in Computer Science, pages 186–197. Springer, 2008.

[91] Mauricio Marin, Veronica Gil-Costa, and Roberto Uribe. Hybrid index for metric space databases. In *Proceedings of the International Conference on Computational Science*, 2008.

[92] María Luisa Micó, José Oncina, and Rafael C. Carrasco. A fast branch & bound nearest neighbour classifier in metric spaces. *Pattern Recognition Letters*, 17(7):731–739, June 1996.

[93] Ken Tokoro, Kazuaki Yamaguchi, and Sumio Masuda. Improvements of TLAESA nearest neighbour search algorithm and extension to approximation search. In *Proceedings of the 29th Australasian Computer Science Conference*, pages 77–83. Australian Computer Society, Inc., 2006.

[94] Frank K. H. A. Dehne and Hartmut Noltemeier. Voronoi trees and clustering problems. *Information Systems*, 12(2):171–175, 1987.

[95] Peter N. Yianilos. Data structures and algorithms for nearest neighbor search in general metric spaces. In *Proceedings of the fourth annual ACM-SIAM Symposium on Discrete algorithms*, pages 311–321, Philadelphia, PA, USA, 1993. Society for Industrial and Applied Mathematics.

[96] Vladimir Pestov and Aleksandar Stojmirović. Indexing schemes for similarity search : an illustrated paradigm. *Fundamenta Informaticae*, 70 (4):367–385, 2006.

[97] Edgar Chávez and Gonzalo Navarro. Metric databases. In Laura C. Rivero, Jorge Horacio Doorn, and Viviana E. Ferraggine, editors, *Encyclopedia of Database Technologies and Applications*, pages 367–372. Idea Group, 2006.

[98] Tomáš Skopal. *Metric Indexing in Information Retrieval.* PhD thesis, Technical University of Ostrava, 2004.

[99] Elena Deza and Michel Marie Deza. *Dictionary of Distances.* Elsevier, 2006.

[100] Magnus Lie Hetland. A survey of recent methods for efficient retrieval of similar time sequences. In Mark Last, Abraham Kandel, and Horst Bunke, editors, *Data Mining in Time Series Databases*, chapter 2, pages 23–42. World Scientific, 2004.

[101] Claudio Gennaro, Pasquale Savino, and Pavel Zezula. Similarity search in metric databases through hashing. In *Proceedings of the 2001 ACM workshops on Multimedia: multimedia information retrieval*, 2001.

[102] Stephen Semmes. What is a metric space? arXiv:0709.1676v1 [math.MG], September 2007.

[103] Pawan K. Jain and Khalil Ahmad. *Metric Spaces.* Alpha Science International Ltd., Pangbourne, U.K., second edition, 2004.

[104] Will Archer Arentz, Magnus Lie Hetland, and Bjørn Olstad. Methods for retrieving musical information based on rhythm and pitch correlations. *Journal of New Music Research*, 34(2), 2005.

[105] Dieter Jungnickel. *Graphs, Networks and Algorithms.* Springer, second edition, 2005.

[106] Hartmut Noltemeier. Voronoi trees and applications. In *Proceedings of the International Workshop on Discrete Algorithms and Complexity*, pages 69–74, 1989.

[107] Yannis Manolopoulos, Alexandros Nanopoulos, Apostolos N. Papadopoulos, and Yannis Theodoridis. *R-Trees : Theory and Applications*. Advanced Information and Knowledge Processing. Springer, 2006.

[108] M. Schroeder. *Fractals, Chaos, Power Laws*. W. H. Freeman and Company, sixth edition, 1991.