Analyzing Metric Space Indexes: What For?

(Invited Paper)

Gonzalo Navarro⋆
Dept. of Computer Science
University of Chile
Santiago, Chile
gnavarro@dcc.uchile.cl

***Abstract*—It has been a long way since the beginnings of metric space searching, where people coming from algorithmics tried to apply their background to this new paradigm, obtaining variable, but especially difficult to explain, success or lack of it. Since then, some has been learned about the specifics of the problem, in particular regarding key aspects such as the intrinsic dimensionality, that were not well understood in the beginning. The inclusion of those aspects in the picture has led to the most important developments in the area.**

Similarly, researchers have tried to apply asymptotic analysis concepts to understand and predict the performance of the data structures. Again, it was soon clear that this was insufficient, and that the characteristics of the metric space itself could not be neglected. Although some progress has been made on understanding concepts such as the curse of dimensionality, modern researchers seem to have given up in using asymptotic analysis. They rely on experiments, or at best in detailed cost models that are good predictors but do not explain why the data structures perform in the way they do.

In this paper I will argue that this is a big loss. Even if the predictive capability of asymptotic analysis is poor, it constitutes a great tool to understand the algorithmic concepts behind the different data structures, and gives powerful hints in the design of new ones. I will exemplify my view by recollecting what is known on asymptotic analysis of metric indexes, and will add some new results.

I. Introduction

My first contact with the metric space search problem, back in 1997, was a paper by my advisor Ricardo Baeza-Yates, Walter Cunto, Udi Manber, and Sun Wu, about "Fixed-Queries Trees" (FQTs), in *CPM'94* [1]. The paper was about modifying Burkhard-Keller Trees (BKTs) [2], possibly the oldest data structure for this problem (1973).

In the so-called *metric space approach* one had a database U of objects belonging to a metric space (X, d), where $d : X \times X \to \mathbb{R}$ was a *distance function* satisfying the triangle inequality. The objects were black boxes, so the only possible action upon them was to compute pairwise distances. The goal was to build data structures on U so that later, given a query ball $(q, r) \in X \times \mathbb{R}^+$, one could return all the objects of U within distance r to q. This is called a "range query". Other more sophisticated ones, such as returning the k nearest neighbors, were also of interest. The complexity measure was just the number of distance computations, as they were supposed to dominate the running time.[1]

(⋆) Partially funded by Millennium Institute for Cell Dynamics and Biotechnology (ICDB), Grant ICM P05-001-F, Mideplan, Chile.

Apart from being such an elegant abstraction of many seemingly unrelated problems, what fascinated me about this metric space approach was the algorithmic structure that was uncovered. Unlike trees for exact searching, one had to enter into several branches of these trees, which turned the typical $O(\log n)$-time exact search into an intriguing $O(n^\alpha)$-time search, for some $0 \leq \alpha \leq 1$ that depended on the problem in an obscure way. This kind of search was not shattering news in the data structures world. It was well known in multidimensional data structures [6], for example. Yet, it was definitely not the classical exact searching using trees.

There were by then many other data structures designed for metric space searching. Most were tree data structures of one kind or another [7]–[14], and more trees came later [14]–[17].[2]

As in any algorithmic development, the concern of how to measure the performance was raised. Experiments are of course one choice, but they have well-known shortcomings (cost, difficulty to extrapolate, non-explanatory results, etc.). In this case, an extra problem was that results were very different depending on the metric space considered, and that there was not (and there is not today) agreement on which could be representative metric spaces, or how the results in one space could help predict the performance in others.

In the early days of computing, asymptotic analysis was invented to overcome similar problems. It not only gives a measure of performance that is basically technology-independent, explanatory, extrapolation-friendly, and that can be obtained without experiments, but also it interacts strongly with the design of the algorithm, so that analysis can help improve the design and even suggest radically new solutions, not only predict the growth rate of the cost for large inputs.

[1]Beware: this paper is not intended to be a survey on the topic! If needed, there are exhaustive surveys and books [3]–[5].

[2]I am mentioning just a sample that includes representatives of the distinct approaches I want to discuss in this paper. My recollection is not meant to be complete in any way.

It is not surprising, then, that researchers tried to apply asymptotic analysis to their metric space data structures. One can see such attempts since the early papers [1], [2], [7]–[10], [15], [18]. In my own experience, however, these attempts to understand how were those structures going to perform asymptotically were almost always poor predictors of the actual performance of the indexes. There were too many factors, apart from the known imprecision of asymptotic analysis, that impacted performance. It is not that the growth rate figures do not show up in the experiments, but rather that the constant factors are very hard to predict, as they depend in a very complicated way on the structure of each metric space. So far, the only way to tell which structure will perform better in a given metric space is experimentation.

Several other researchers in the past seem to have reached a similar conclusion, because one can identify a slowly emerging concern about the specific nature of the metric indexing problem. Several papers [11]–[13], [16], [19], [20] started to wonder about topics that strongly impacted performance in practice, and to which previous asymptotic analyses were blind, such as how to choose the reference-point objects, or why some metric spaces seemed to be intrinsically more difficult to index than others (this was to be called "the curse of dimensionality").

Authors like Yianilos and Brin have to be commended for having led the most serious effort I am aware of to combine asymptotic analysis with concerns that are specific of metric spaces. Their analyses led to the design of data structures such as the "Vantage Point Forest" [21], which offers worst-case asymptotic performance guarantees (a very uncommon achievement). Still, the predictive power of such analyses is very limited.

From some point, researchers seem to have given up and to resort almost exclusively to experiments [12], [14], [17]. Others developed very detailed cost models, which were successful in practice to predict performance, but they had to be computed on a given instance of the data structure [22], [23]. This made them good tools for tuning a data structure, but not to understand the performance of the general idea. Asymptotic analysis of metric indexes seems to have been abandoned. I have not seen one in *SISAP 2008*, and by the time of writing this paper, I am not aware of any in *SISAP 2009*. Indeed, except for my own extemporaneous attempts [15], [16], I do not remember having seen any of those asymptotic analyses in this millennium.

Other types of analyses, focusing on understanding the curse of dimensionality, have made more progress, fortunately. An example can be seen in this very same *SISAP* [24]. This topic by itself would deserve a whole separate article, so I will not attempt to cover the work that has been done around it.

My opinion is that having abandoned asymptotic analysis is a great loss. It is clear that it is not useful for predicting the actual performance of indexes, for comparing indexing methods, or for automatic fine-tuning of data structures. But I strongly believe that asymptotic analysis is an excellent tool for *understanding* why the indexes behave in the way they do, for *guiding the design* of new indexes, and for understanding *in which direction* could they be optimized. As I understand it today, asymptotic analysis does not serve at all as a *quantitative* tool for predicting performance, but rather as a *conceptual* tool for the design and improvement of metric indexes. Questions such as "why should a tree be balanced?", "what can I expect from increasing its arity?", "how should the space be partitioned?", "which is the space/time tradeoff of pivots?", "what to expect if the histogram shrinks?", among many others, are conceptual in nature, and may lead to the development of new indexing techniques. The associated quantitative questions must be answered through experiments or detailed cost models, and may lead to practical improvements on the existing data structures.

Let me now, first, try to justify my thesis via a short detour to my own experiences on asymptotic analysis and design of metric indexes. Then I will switch to a more mathematical exposition, trying to collect what is known on asymptotic analysis of the most typical metric indexing approaches, while adding some new results.

II. An Algorithmicist in Wonderland

Almost simultaneously to my discovery of the area, I met Edgar Chávez, a Mexican mathematician doing his PhD on this topic. Although my own PhD topic was only marginally related, we started to work together in what has turned out to be my longest-lasting collaboration with a colleague. We found several early approaches to the problem, such as Bisector Trees (BSTs, 1983) [7], Voronoi Trees (VTs, 1987) [8], Generalized Hyperplane Trees (GHTs, 1991) [9], [10], Vantage-Point Trees (VPTs, 1993) [9], [11], Geometric Near-Neighbor Access Trees (GNATs, 1995) [12], Multi VPTs (MVPTs, 1997) [13], M-trees (1997) [14], and Vantage-Point Forests (VPFs, 1999) [21], just to name the then-most prominent ones.

It seemed to be all about trees. MTs, VPTs, MVPTs, and VTs were balanced trees. BKTs, FQTs, BSTs, GHTs, and GNATs were not, but the intention of the authors was not to have unbalanced trees, rather it was a kind of undesirable effect. So this seemed to be a well-behaved branch of classical algorithmics. Only VPFs seemed not to fit well, as they could be been seen as on-purpose unbalanced trees (despite that the author preferred to regard them as a forest of balanced trees). The more difficult the problem, the more and smaller trees were needed to obtain the best performance.

By then, I came across a pretty undecipherable paper by Faragó, Linder and Lugosi [25], which I failed to parse after several attempts. Fortunately Edgar, with more background on that kind of mathematics, and enthusiastic about the paper, preprocessed it for me. The paper was about how to optimally choose pivots in a space, proving that the best was to form a kind of convex hull around the data points with the pivots. Intriguingly, it promised $O(1)$ search time, which looked as a heresy to me. But there was also the old work by Vidal [26] (1986) showing the same $O(1)$ time in practice (yet the constant depended on the characteristics of the metric space). After all, a single distance computation gives much more information than a binary value, so there was no reason for $\Omega(\log n)$ to be a complexity barrier. Thus, basic assumptions on trees seemed to be inadequate.

Most intriguingly about Vidal's work (AESA) and the subsequent LAESA [27] (1994), was that it was not at all about data structures. There were no trees! Distances to some elements were simply tabulated, and the most possible information from those distances and the triangle inequality was used to avoid distance computations. It had always been said that extra CPU costs apart from computing distances were going to be ignored, but this time they meant it: In AESA, there was $O(n)$ extra CPU time per each new distance computed. The core of the technique was about how to choose the next pivot. This was a heuristic chosen with some spatial arguments. Adding more to the heresy, AESA simply stored the $\Theta(n^2)$ distances among all database objects, so the space and preprocessing time was an issue. It was clear that the algorithmics were just a part of the story.

This intriguing duality between the algorithmics and the metric-space specifics, and their poorly understood interrelations, motivated Edgar and I to write a survey on the topic in *ACM Computing Surveys* with our advisors [3]. Our first motivation was to understand ourselves what was going on. Our second motivation was to collect the many unrelated works that we had found, which rediscovered the same ideas in the different application areas.

Although that, once essential, survey is now superseded by more modern and comprehensive books [4], [5], and the second motivation is now better addressed with *SISAP* conference, I still believe this survey was an important milestone to warn the algorithmicists (or maybe just ourselves) that they were missing which was probably the main point: Metric space indexing was not much about data structures. It was more about storing information on intra-database distances, so that at query time one should extract up to the last drop of knowledge from that information, in order to save every possible distance computation, and pay the cost for the rest. The extra CPU time was irrelevant (which is, admittedly, a strong simplifying assumption in practice, allowing serious abuse). Trees were no more than a convenient way of storing and factoring out partial information on distances (and also saving some extra CPU time, by the way). The main point to design a metric indexing approach was which information to store (which distances, how much precision, etc.). The curse of dimensionality and the histograms of distances played a central role in explaining why a metric space was harder to search than others, and to choose which information was best to store.

This point of view, albeit somewhat extreme, was enormously successful. It opened the door (to us, and presumably to others) to understanding that the coin had another very important side, and that this was not yet another field where known algorithmic concepts could be directly applied.

The design of our List of Clusters (LC) [16] proved to be particularly revealing. In the LC, each cluster has a center and only the distances from the center to its cluster elements are stored, that is, we store just one short distance per element. The search must sequentially consider each center, and discard the cluster or not according to the distance between the query and the center. The algorithmic concepts behind the LC were trivial. Yet, in high-dimensional spaces the LC was virtually unbeatable, whereas all trees and fancy data structures failed catastrophically.

Still, there was an intriguing aspect. The LC could also be seen as an extremely unbalanced MT or VPT! This led us back to the beginning. The earliest authors strove to achieve balanced trees. We had learned that, somehow, this was not the point. Now it turned out that, for higher dimensionalities, unbalanced trees were preferable. Moreover, we could see experimentally that, the higher the dimensionality, the more unbalanced should be the tree (that is, more and smaller clusters) to achieve optimum performance (and this optimum performance was worse anyway, hence the curse of dimensionality). This shed a new light on other data structures, such as the VPF. We attempted a relatively coarse analysis of LCs to try to explain what was going on [16], which followed the trend of similar earlier attempts [1], [11], [21]. Essentially, the result was again the old $O(n^\alpha)$ time complexity, but now we obtained more insight into the relation between α and the properties of the metric space.

In broad terms, our analysis explained the relation between dimensionality and unbalancing in tree data structures, somehow hinted but, in my opinion, not fully exposed by Yianilos [21]. I believe that analysis is indeed inspiring, and explains in part the interrelation between the algorithmic approach to the problem and the specifics of metric space searching. Again, it turned out to be useless for predicting performance or for finding the optimal cluster sizes, but it was key to understand the critical point of balancing, which we have had in mind in much of our later work.

I will start the formal part of the paper by reviewing and extending these previous analyses on pivot tables and pivot-based trees [1], [3], [11], [16], [21], [28], showing that they apply to a wide class of indexes. I will then give new analyses for clustering trees via ball partitioning and via hyperplane partitioning, using my own previous work on Spatial Approximation Trees (SATs) as inspiration for the latter [15]. The analysis for clusters is brand new in metric spaces; some analogies with previous ones in vector spaces exist [29] (thanks to Benjamin Bustos for this pointer).

III. Pivot Tables

Possibly the simplest metric data structure is the pivot table. Pivots $c_1 \ldots c_k \in U$ are chosen, and all the distances $d(u, c_i)$ are stored in a table for all $u \in U$, $1 \leq i \leq k$. Upon a range query (q, r), q is compared with each c_i, which is reported if $d(q, c_i) \leq r$, and any $u \in U$ such that $|d(q, c_i) - d(u, c_i)| > r$ is discarded thanks to the triangle inequality. All the rest are directly compared with q.

Let us call $f(x)$ the histogram of distances of the metric space. If the c_is are chosen at random, then $f(x)$ is also the histogram of distances to any c_i. Let us call $F(x)$ the accumulated histogram. If $d(q, c_i) = x$ (which has probability $f(x)$ for a random q), then the probability that c_i does not discard a random $u \in U$ is that of $d(u, c_i)$ belonging to the interval $[x - r, x + r]$ of the histogram. Then the probability that a random $q \in X$ does not discard a random $u \in U$ is

$$e \;=\; \int_0^{+\infty} f(x) \cdot (F(x+r) - \lim_{z \to 0^+} F(x - r - z)),$$

where $0 \leq e \leq 1$. Note that, the larger r or the more concentrated $f(x)$, the closer e to 1. Now, if pivots c_i are chosen randomly, the probability that u is not discarded is e^k, and thus the expected cost of the search is

$$T(n) \;=\; k + ne^k.$$

This is where the asymptotic and the "engineering" analyses differ. The latter starts from here to devise methods to choose good pivots [23], or to predict the precise pivot performance [22], from a histogram of distances. Our goal is different, almost orthogonal. We want to understand the general asymptotic behavior of the technique, without caring much about the details and constants.

Interestingly, it is possible to achieve logarithmic search time using sufficiently many pivots. With $k^* = \log_{1/e} n + \log_{1/e} \ln(1/e)$ pivots, the expected search time is

$$T(n) \;=\; \log_{1/e} n + \frac{1}{\ln(1/e)} \;=\; O(\log n),$$

where we can note that the constants worsen as e tends to 1. These constants, but not the asymptotic behavior, depend on the metric space and the search radius. In practice, however, k^* can be unreachable for memory limitations (recall that k^* also grows as e tends to 1), and thus it might be that we can only pay for $k = \beta \ln n$ pivots. In this case the cost becomes

$$T(n) \;=\; \beta \log n + ne^{\beta \ln n} \;=\; O\left(n^{1 - \beta \ln(1/e)}\right),$$

which is of the form n^α for some $0 \leq \alpha \leq 1$ that depends on $f(x)$, on r, and on the available memory. Finally, if we can afford only a fixed number of pivots, then the search time is $O(n)$, with a constant depending on k, $f(x)$, and r.

In any case, note that we will always compare the, on average, $nF(r)$ elements that must be returned. Our analysis here refers to discarding *random* elements u. Those close to q are not random and the analysis does not apply to them. We will omit mentioning this point again in the next sections.

Recall that more sophisticated analyses [25] establish that the time can indeed be $O(1)$ if the pivots are chosen in a certain way, whereas we see that by choosing them at random this is not possible. This encourages the quest for good pivot selection methods. Yet, Faragó et al. [25] make some obscure assumptions on the nature of the metric space, which are not easy to verify. On the other hand, some experimental results that seem to verify this constant-time cost are given by Vidal [26]. In that case, however, the pivots are chosen *for each query*, which is not possible unless a large number of them (or all) are indexed. An intermediate technique, using a large set of pivots and then choosing the best ones for each query, is LAESA [27].

IV. Trees for Pivots and Rings

Consider a tree structure such that an element $c \in U$ is chosen for the root, and each child represents a range of distances to c. That is, the root has k children $T_1 \ldots T_k$, representing ranges $(\ell_i, \ell_{i+1}]$, such that $\ell_1 = 0$ and $\ell_{k+1} = +\infty$. Subtree T_i will contain all the $v \in U$ such that $d(v, c) \in (\ell_i, \ell_{i+1}]$ (i.e., a ring around c), and will be organized recursively. When searching for a ball (q, r), q is compared with c and c is reported if $d(q, c) \leq r$. Then, the search recursively enters each T_i such that $[d(q, c) - r, d(q, c) + r] \cap (\ell_i, \ell_{i+1}] \neq \emptyset$. The others can be discarded thanks to the triangle inequality. This general formulation encompasses the essentials of a number of indexes, including BKTs, MTs, FQTs, VPTs, MVPTs, VPFs and LCs.

If c is chosen at random, then the global histogram $f(x)$ is also the histogram of distances to c. Let us call $p_i = F(\ell_{i+1}) - F(\ell_i)$ the probability that a random element falls within the interval corresponding to subtree T_i, and $q_i = F(\ell_{i+1} + r) - F(\ell_i - r) \geq p_i$ the probability that a search for a random q with radius r must enter subtree T_i. Assume for simplicity that the structure maintains recursively the same p_i values within the subtrees (as in MTs, VPTs, MVPTs, and some variants of LCs). Assume further that the q_i values remain similar within subtrees (that is, the histograms $f_i(x)$ within subtrees T_i are similar to the global histogram[3]). Albeit these probabilities are not independent, still the expectations commute with the sum. Thus the expected number of distance computations for a random query $q \in X$ satisfies the recurrence

$$T(n) \;=\; 1 + \sum_{1 \leq i \leq k} q_i \; T(p_i n), \quad T(1) \;=\; 1, \qquad (1)$$

whose solution, easily verifiable by induction, is

$$T(n) \;=\; \frac{sn^\alpha - 1}{s - 1} \;=\; O(n^\alpha), \quad s \;=\; \sum_{1 \leq i \leq k} q_i, \quad (2)$$

[3]This is, admittedly, a gross simplification, which ruins the predictive power of the analysis. Yet, this is useful to derive conceptual conclusions.

where α is the solution of the equation

$$\sum_{1 \le i \le k} q_i p_i^\alpha \;=\; 1. \qquad (3)$$

It is not hard to see that there is a unique solution for α in $[0,1]$: As α moves continuously from 0 to 1, the sum (3) goes from above to below 1 with negative derivative. The solution is 0 only if $\sum q_i = 1$, which means $r = 0$ (i.e., exact search) unless the histogram is rather particular. Actually, if $r = 0$ (and thus $q_i = p_i$) the solution to the recurrence is $O(\log n)$ as for classical search trees. A positive search radius turns an exact into approximate, and radically changes the behavior of the scheme.

In some variants, like the FQT, the tree root is the same for all the nodes of a given depth (thus the root does not necessarily belong to the set, and a leaf is formed when a sufficiently small number of elements remains in the subtree). Those "roots" are thus better called pivots again, and regarded as external to the tree. The search cost is, on one hand, the number of pivots (or the height of the tree) and, on the other, the same recurrence (1) without the "1+" in the beginning, which accounts for the elements at the reached leaves, as these must be compared to the query. The former is $\log_{1/p_{\min}} n$, where $p_{\min} = \min\{p_1 \ldots p_k\}$, whereas the latter is simply n^α. Thus the solution is of the same order; the difference in constants favors the FQT, which has been verified experimentally [1], [28]. Note also that, even using a large number of pivots, the cost is not logarithmic as in Section III. The reason is that elements are not compared with all the pivots, but tree leaves may be formed before the tree height is reached. When the tree is forced to have all the leaves at the maximum depth (FHQTs) [28], the analysis is similar as with a pivot table.

Let us now focus on the MVPT, which allows several pivots per node (say, t). To search, we compute t distances, and then enter into the children corresponding to the intersections of t ranges our query ball intersects. Then our recurrence (1) would be as follows.

$$T(n) \;=\; t \;+ \sum_{1 \le i_1 \ldots i_t \le k} q_{i_1} \ldots q_{i_t}\; T(p_{i_1} \ldots p_{i_t} n), \qquad (4)$$

whose solution is $T(n) = ((t + s^t - 1)n^\alpha - t)/(s^t - 1)$. Again, the solution is of the same order, but the constant $1 + t/(s^t - 1)$ improves as t grows, which recommends again using the maximum t, as in a FQT. Again, the time drops to logarithmic if we choose sufficiently large $t = \Theta(\log n)$, so that $q_{i_1} \ldots q_{i_t} = O(1)$. This does not show up in our $T(n)$ because the analysis ignored rounding effects, which are irrelevant when t is small enough.

The central difference between trees and pivot tables is that the former guarantee linear space, whereas the latter may achieve logarithmic time. Alternatives such as FQTs and MVPTs are in between; they almost always achieve linear space (but do not guarantee it) and do not achieve logarithmic time (but improve the constant factor). In general, using these trees is justified only if we have little (linear) space for the index. Partial experiments [13] do suggest that using larger t is better than just 2 (MVPTs vs VPTs).

Let us now study the arity of the trees. To gain some intuition, assume that all $p_i = p$ (thus $k = 1/p$) and all $q_i = q = p + \delta$ (that is, a relatively flat histogram, with δ depending on r). Then $s = kq$ and we have a closed form expression for $\alpha = \log_k(kq)$. The solution to recurrence (1) is $T(n) = \frac{kq\, n^{1-\log_k(1/q)} - 1}{kq - 1}$. It is illustrative to write the exponent as $\alpha = \log_k(1 + \delta k)$. The minimum α value, as a function of δ, has no closed form expression, but it is achieved for larger k (that is, higher tree arities) as r grows, and yields also larger values of α. Thus, as the search becomes more difficult, a higher-ary tree is more advisable: Even if we have to enter more branches to cover the same range, we have a higher chance of discarding the few ranges that do not intersect the query ball. Again, partial experiments [13] suggest that using higher arities is convenient for higher dimensions.

Finally, let us consider the problem of balancing. Assume we fix an arity k, and wonder which is the best partitioning. To gain intuition, assume again $q_i = p_i + \delta$ for all i, thus we seek to find the p_i values that minimize the α that solves $\sum p_i^\alpha (p_i + \delta) = 1$. The solution where all $p_i = p$ yields the equation $kp^\alpha(p + \delta) = k^{1-\alpha}(1/k + \delta) = 1$, with solution $\alpha = \log_k(1 + \delta k)$, as we have seen. If we perturb just two p_i values, adding and subtracting ϵ, the new equation is $(k - 2)p^\alpha(p + \delta) + (p + \epsilon)^\alpha(p + \epsilon + \delta) + (p - \epsilon)^\alpha(p - \epsilon + \delta) = 1$. Implicit differentiation reveals that α increases as ϵ deviates from zero. Also, intuitively, the function is convex (even with $\delta = 0$) with respect to α, so moving ϵ increases it, thus to retain equality to 1 we have to increase α. This shows that, with a flat histogram, the best is a balanced partition.

This is not anymore the case on higher dimensions. Let us assume now that the histogram corresponds to dimension D, that is, $F(x) = x^D$, with $x \in [0,1]$. To simplify further, assume $k = 2$, so we want the x such that $p = F(x)$ minimizes the α that solves $F(x)^\alpha F(x + r) + (1 - F(x))^\alpha (1 - F(x - r)) = 1$. We can use implicit differentiation to obtain the optimum α as a function of x and r, but again there is no closed-form expression. Already low dimensions such as $D = 2$, for retrieving 0.01% of the data set, recommend leaving $p_1 = 0.24$ of the histogram to the left and $p_2 = 0.76$ to the right, so as to obtain $\alpha = 0.035$. As we let D grow, the analysis recommends more unbalanced partitions to obtain the optimum α, which is nevertheless higher. For example for $D = 6$ it recommends leaving just $p_1 = 6.4 \times 10^{-5}$ of the data on the left child, to achieve $\alpha = 0.522$, and for $D = 10$ it leaves $p_1 = 2.0 \times 10^{-8}$ of the data to the left, to achieve $\alpha = 0.701$. If our query returns 1% of the data, the suggested partitions are even more unbalanced and the achieved αs are larger.

Although the recommendation makes sense in principle, because an unbalanced tree is equivalent to having many pivots per element within linear space, we have to take these numeric recommendations with a grain of salt. On those extremely unbalanced trees one has a high chance (1, for sufficiently large r) of traversing all the right path, whose length is $\log_{1/p_2} n$. This is much larger than the solution we achieve for $T(n)$. This contradiction is explained, again, because our general solution (2) ignores integer rounding effects, which is noticeable when there are extremely small probabilities involved.

This shows, once more, that although asymptotic models are poor as predictors, we can still learn concepts from them, such that unbalancing is preferable for higher dimensions. We have verified this fact experimentally [16].

V. Trees for Centers and Clusters

Trees formed by clustering operate differently. They choose k centers $c_1 \ldots c_k$, with corresponding subtrees $T_1 \ldots T_k$, and insert the other elements into those subtrees with some criterion trying that each T_i is spatially compact (for example, each element is inserted into the tree of the c_i closest to it). Covering radii $cr_i = \max_{u \in T_i} d(c_i, u)$ are computed for each T_i, and the construction proceeds recursively inside each T_i. When querying for a ball (q, r), q is compared with each c_i, reporting it if $d(q, c_i) \leq r$, and entering recursively into T_i if $d(q, c_i) \leq cr_i + r$. This description encompasses the BST, the M-tree, Antipole Trees [17], and variants of the LC.

Assume the centers are chosen at random, thus $p_i = F(cr_i)$ is the probability of a random element falling within the covering radius distance to any c_i, and $q_i = F(cr_i + r)$ is that of a query (q, r) entering T_i. If the cluster balls cover the queriable space, it must hold $\sum_{1 \leq i \leq k} p_i \geq 1$, where equality would hold only if the balls have no intersection (which is usually impossible to achieve, but can be approached via a good clustering method).

Assume for simplicity that we structure the subtrees as clusters of roughly similar probability mass, so that all $p_i = p$, thus $k \geq 1/p$. Assume further that the arity and relative mass of the clusters is maintained in the subtrees, thus the probability mass of a cluster at depth h of the tree is p^h. If the elements are inserted at random in any suitable cluster, then the expected height of the tree is $\log_{1/p} n$. If, instead, we manage to balance the number of elements in the clusters, then the height is smaller, $\log_k n$. (Note that the goals of having similar number of elements and having similar masses can be conflicting.) Thus the height h^* lies between these two values on average.

At depth $h \leq \log_k n$ there will be k^h clusters, whereas this will be at most n for larger h. The probability of the query ball intersecting a cluster at depth h is $F(x_h + r)$, where x_h is the covering radius at level h, that is, $F(x_h) = p^h$. Thus the overall query cost is

$$T(n) \;=\; \sum_{h=1}^{h^*} \min(n, k^h) F(F^{-1}(p^{h-1}) + r).$$

To gain intuition, assume a flat histogram in [0,1], so $F(x) = x$. In this case $F(F^{-1}(p^{h-1}) + r) = p^{h-1} + r$, and thus $T(n) = \sum_{h=1}^{h^*} \min(n, k^h)(r + p^{h-1})$. In the unlikely case of a perfect clustering ($kp = 1$) this is $T(n) = k \log_k n + \frac{k}{k-1}(n-1)r$. In the more likely case $kp > 1$, it is

$$\begin{aligned}
T(n) &= \sum_{h=1}^{\log_k n} k^h(p^{h-1} + r) + n \sum_{h=\log_k n+1}^{h^*} (p^{h-1} + r) \\
&= \frac{k(n^{1-\log_k(1/p)} - 1)}{kp - 1} + \frac{k}{k-1} r(n-1) + \\
&\quad \frac{1 - p^{h^* - \log_k n}}{1 - p} n^{1-\log_k(1/p)} + (h^* - \log_k n)\, rn \\
&\leq \frac{k-1}{(kp-1)(1-p)} n^{1-\log_k(1/p)} + \\
&\quad \left(\frac{k}{k-1} + h^* - \log_k n\right) rn.
\end{aligned}$$

Note that rn is in this case the expected output size, and thus it corresponds to the part of the cost we do not count as we take it as unavoidable. There is, however, a multiplicative overhead on that part of the cost, which improves for larger k (making a difference only for small k, actually), and for better balanced trees (that is, those that have about the same number of elements per child). The other part of the cost is, again, of the form $O(n^\alpha)$, and if we take kp as a factor that depends on the amount of overlapping more than on k, we have that the exponent improves as k grows ($\alpha = \log_k(kp)$), while the constant increases. The constant also increases if the clusters cover much of the space (which is alleviated also by using more clusters)

Thus it is clearly advantageous to have a good clustering of the space, so that the probability of falling in the overlap of two clusters is minimized (which reduces kp), and, particularly when the search is more difficult, to have children clusters of about the same number of elements (so that the tree is more balanced and $h^* - \log_k n$ is reduced). Those are clear design goals in the M-tree, for example. It is also interesting that the issue of balancing has reappeared, now giving advantage to balancing trees managing clusters.

With respect to the best k, it is in principle better to have a larger k, especially when the search is more difficult. In the easier spaces, however, using a very large k might have a pernicious effect in performance because of the constant k multiplying the $O(n^\alpha)$ term, especially if the overlaps are significant and thus $1/p$ is not too far from 1. LCs are a good example showing how the use of a very large k (so large that the hierarchy actually disappears) is indeed convenient, and higher k is better when the search problem is harder. This has been clearly demonstrated by experiments [16].

If we wish to generalize to dimension D, $F(x) = x^D$, then we have $T(n) = \sum_{h=1}^{h^*} \min(n, k^h)(r + p^{(h-1)/D})^D$. Since $r^D + p^{h-1} \leq (r + p^{(h-1)/D})^D \leq 2^D(r^D + p^{h-1})$, the result is conceptually the same as for $D = 1$.

VI. Trees for Hyperplanes and Voronoi Regions

These trees also have centers $c_1 \ldots c_k$ at each node, and the elements are always sent to the subtree of their closest center. The difference is that the search for a ball (q, r) enters into the subtree T_i such that c_i is the closest center to q, and also to any T_j such that $d(q, c_j) - d(q, c_i) \leq 2r$. This corresponds to dividing the regions using hyperplanes rather than balls, so that the centers induce a Voronoi-like partitioning of the space, and is used in VTs, GHTs, GNATs, and SATs. Regions do not overlap in this scheme.

The analysis of the SAT [15] is useful for this case. Let $D_i = d(q, c_i)$ and $G_i = D_i - \min(D_1, \ldots, D_k)$. Let $g_i(x)$ be the histogram of G_i (which has mass zero over $x < 0$) and $G_i(x)$ the cumulative function of $g_i(x)$. Then the probability g_i of entering T_i is that of $G_i \leq 2r$, that is, $g_i = G_i(2r)$. Hence the recurrence for the search cost is

$$\begin{aligned} T(n) &= k + \sum_{1 \leq i \leq k} p_i \left(T(p_i n) + \sum_{j \neq i} g_j \, T(p_j n) \right) \\ &= k + \sum (p_i + g_i(1 - p_i)) \, T(p_i n), \end{aligned}$$

where again p_i is the probability of being closer to c_i than to any other center, and again we have assumed that the g_is stay similar within each cluster. The solution is

$$T(n) \;=\; \frac{(k + s - 1)n^\alpha - k}{s - 1}, \quad s \;=\; 1 + \sum_{1 \leq i \leq k} g_i(1 - p_i),$$

where α is the solution of

$$\sum_{1 \leq i \leq k} (p_i + g_i(1 - p_i)) \, p_i^\alpha \;=\; 1.$$

It can be seen that, for $\alpha = 1$, this sum is ≤ 1, as it increases with the g_is and reaches 1 when all $g_i = 1$. For $\alpha = 0$, it is ≥ 1, as it reaches 1 when all $g_i = 0$. As r grows, the g_is grow and so does α.

If the partitions are similar, so that $p_i = 1/k$, then $\alpha = \log_k s$. Assume also $g_i = p_i + \delta$, so $s = 1 + k(p + \delta)(1 - p) = 2 - 1/k + (k - 1)\delta$. The optimum α as a function of k, again, has no closed form expression, but one can find it numerically, to verify that the best k grows with δ (that is, with r and with the dimensionality). Another subtler effect is that δ also depends on k: As k grows, $\min(D_1, \ldots, D_k)$ decreases, G_i increases, and $g_i = G_i(2r)$ decreases, so does δ. Yet, note that $G_i(2r) \geq F(2r)$, thus for large enough k $G_i(2r)$ gets close to $F(2r)$ and this effect is less noticeable. Indeed, it was already shown with GNATs versus GHTs that using higher arities was beneficial up to some point [12]. Also, SATs perform better with higher arities when the problem is harder [15].

Although these trees are not naturally unbalanced, balancing is difficult to ensure. On the other hand, the formulas do not suggest any particular effect in this case. With respect to covering similar probability masses with the children of a node, we can do as in Section IV and assume all $p_i = p = 1/k$ (and $g_i = g$) except for two, which are $p + \epsilon$ and $p - \epsilon$ (and g^+ and g^-). Writing $p + g(1 - p) = p(1 - g) + g$, we have $(k - 2)p^\alpha(p(1 - g) + p) + (p + \epsilon)^\alpha((p + \epsilon)(1 - g^+) + g^+) + (p - \epsilon)^\alpha((p - \epsilon)(1 - g^-) + g^-) = 1$. Noting that $(p(1 - g) + g)$ is a convex combination (with weight $0 \leq g \leq 1$) between p and 1, we have again that this is a convex function of ϵ, and thus α increases as ϵ moves away from zero, thus balancing the probability masses is better.

VII. Conclusions

In this paper I have argued that asymptotic analysis of metric spaces, although normally hopeless for predicting detailed performance and for fine-tuning data structures, is extremely useful to understand why they behave as they do, and to give powerful hints in the design of new data structures. I have shown how this has aided in my own understanding of the area and have given a number of (old and new) analytical results that help understand the key concepts of most of the data structures available today.

As an example, I have shown novel analytical arguments in favor of balancing clustering trees, thus giving an analytical justification for balanced data structures such as the M-tree [14]. In this sense I feel our previous comment [3] about the convenience of balancing in the case of secondary memory data structures missed the point. Balancing the M-tree is good because it is a clustering tree, not because it is in secondary memory.

Indeed, secondary memory is an almost orthogonal issue, so all our analyses should apply there. I believe the main point for secondary memory is not to design special data structures for that case, but to adapt main-memory ones so that the *packing* of data into blocks is optimized, both in the sense of minimizing the wasted space and of trying to put together data that will be likely accessed together, so as to minimize the ratio of I/Os versus distances computed. A good example of these concerns is our recent design of a secondary-memory SAT, in this very same *SISAP* [30].

With respect to the analyses, perhaps the least satisfactory aspect of the results is the inability to compare different types of indexing structures. In all realistic cases the costs are of the form $O(n^\alpha)$, but the α values depend on many factors, including especially the structure of the metric space. Extending these results to be able of comparing across approaches would be a great success for the analysis, as it could be able of recommending which approach to use depending on the characteristics of the metric space.

There are several other extensions of the problem I have not considered. One is nearest neighbor searching, for which the development of a general range-optimal strategy [4], [10] lets one reduce the analysis to that of range searching. A second is approximate algorithms [31] (that is, which can err in giving the exact answer), where a few beautiful analyses exist, most notably that of Clarkson [18]. In this case the analyses have focused more on the space than on the asymptotics, and it would be interesting to see if some relevant asymptotic results can be proved. Finally, I believe that dynamic data structures for metric spaces, a clearly undeveloped area, could demand novel asymptotic analyses, and benefit from the resulting suggestions.

References

[1] R. Baeza-Yates, W. Cunto, U. Manber, and S. Wu, "Proximity matching using fixed-queries trees," in *Proc. 5th CPM*, ser. LNCS 807, 1994, pp. 198–212.

[2] W. Burkhard and R. Keller, "Some approaches to best-match file searching," *CACM*, vol. 16, no. 4, pp. 230–236, 1973.

[3] E. Chávez, G. Navarro, R. Baeza-Yates, and J. Marroquín, "Searching in metric spaces," *ACM Comp. Surv.*, vol. 33, no. 3, pp. 273–321, 2001.

[4] H. Samet, *Foundations of Multidimensional and Metric Data Structures*. Morgan Kaufmann, 2005.

[5] P. Zezula, G. Amato, V. Dohnal, and M. Batko, *Similarity Search: The Metric Space Approach*, ser. Advances in Database Systems. Springer, 2006, vol. 32.

[6] P. Agarwal and J. Erickson, "Geometric range searching and its relatives," in *Advances in Discrete and Computational Geometry*. AMS Press, 1999, pp. 1–56.

[7] I. Kalantari and G. McDonald, "A data structure and an algorithm for the nearest point problem," *IEEE Trans. Softw. Eng.*, vol. 9, no. 5, 1983.

[8] F. Dehne and H. Noltemeier, "Voronoi trees and clustering problems," *Inf. Sys.*, vol. 12, no. 2, pp. 171–175, 1987.

[9] J. Uhlmann, "Implementing metric trees to satisfy general proximity/similarity queries," in *Proc. Command and Control Symposium*, Washington, DC, 1991, also Code 5570 NRL Memo Report (1991), Naval Research Laboratory.

[10] ——, "Satisfying general proximity/similarity queries with metric trees," *Inf. Proc. Lett.*, vol. 40, pp. 175–179, 1991.

[11] P. Yianilos, "Data structures and algorithms for nearest neighbor search in general metric spaces," in *Proc. 4th SODA*, 1993, pp. 311–321.

[12] S. Brin, "Near neighbor search in large metric spaces," in *Proc. 21st VLDB*, 1995, pp. 574–584.

[13] T. Bozkaya and M. Ozsoyoglu, "Distance-based indexing for high-dimensional metric spaces," in *Proc. 17th SIGMOD*, 1997, pp. 357–368, Sigmod Record 26(2).

[14] P. Ciaccia, M. Patella, and P. Zezula, "M-tree: an efficient access method for similarity search in metric spaces," in *Proc. 23rd VLDB*, 1997, pp. 426–435.

[15] G. Navarro, "Searching in metric spaces by spatial approximation," *The VLDB Journal*, vol. 11, no. 1, pp. 28–46, 2002.

[16] E. Chávez and G. Navarro, "A compact space decomposition for effective metric indexing," *Patt. Rec. Lett.*, vol. 26, no. 9, pp. 1363–1376, 2005.

[17] D. Cantone, A. Ferro, A. Pulvirenti, D. R. Recupero, and D. Shasha, "Antipole tree indexing to support range search and k-nearest neighbor search in metric spaces," *IEEE Trans. Knowl. Data Eng.*, vol. 17, no. 4, pp. 535–550, 2005.

[18] K. Clarkson, "Nearest neighbor queries in metric spaces." *Discr. Comp. Geom.*, vol. 22, no. 1, pp. 63–93, 1999.

[19] M. Shapiro, "The choice of reference points in best-match file searching," *CACM*, vol. 20, no. 5, pp. 339–343, 1977.

[20] B. Bustos, G. Navarro, and E. Chávez, "Pivot selection techniques for proximity searching in metric spaces," *Patt. Rec. Lett.*, vol. 24, no. 14, pp. 2357–2366, 2003.

[21] P. Yianilos, "Excluded middle vantage point forests for nearest neighbor search," in *DIMACS Implementation Challenge, ALENEX*, 1999.

[22] P. Ciaccia, M. Patella, and P. Zezula, "A cost model for similarity queries in metric spaces," in *Proc. 17th PODS*, 1998, pp. 59–68.

[23] B. Bustos and G. Navarro, "Probabilistic proximity search algorithms based on compact partitions," *J. Discr. Alg.*, vol. 2, no. 1, pp. 115–134, 2003.

[24] I. Volnyansky and V. Pestov, "Curse of dimensionality in pivot-based indexes," in *Proc. 2nd SISAP*, 2009.

[25] A. Faragó, T. Linder, and G. Lugosi, "Fast nearest-neighbor search in dissimilarity spaces," *IEEE Trans. Patt. Anal. Mach. Intell.*, vol. 15, no. 9, pp. 957–962, 1993.

[26] E. Vidal, "An algorithm for finding nearest neighbors in (approximately) constant average time," *Patt. Rec. Lett.*, vol. 4, pp. 145–157, 1986.

[27] L. Micó, J. Oncina, and E. Vidal, "A new version of the nearest-neighbor approximating and eliminating search (AESA) with linear preprocessing-time and memory requirements," *Patt. Rec. Lett.*, vol. 15, pp. 9–17, 1994.

[28] R. Baeza-Yates and G. Navarro, "Fast approximate string matching in a dictionary," in *Proc. 5th SPIRE*, 1998, pp. 14–22.

[29] C. Böhm, "A cost model for query processing in high dimensional data spaces," *ACM Trans. Database Systems*, vol. 25, no. 2, pp. 129–178, 2000.

[30] G. Navarro and N. Reyes, "Dynamic spatial approximation trees for massive data," in *Proc. 2nd SISAP*, 2009.

[31] M. Patella and P. Ciaccia, "The many facets of approximate similarity search," in *Proc. 1st SISAP*, 2008, pp. 10–21.