Information Systems 59 (2016) 15–47

Contents lists available at ScienceDirect

Information Systems

journal homepage: www.elsevier.com/locate/infosys

Faster proximity searching with the distal SAT

Edgar Chávez [a,*], Verónica Ludueña [b], Nora Reyes [b], Patricia Roggero [b]

[a] CICESE, Ensenada, Baja California, Mexico
[b] Departamento de Informática, Universidad Nacional de San Luis, Argentina

ARTICLE INFO

Available online 9 January 2016

Keywords:
Similarity search
Metric spaces
Metric access methods

ABSTRACT

Searching by proximity has been a source of puzzling behaviors and counter-intuitive findings for well established algorithmic design rules. One example is a linked list; it is the worst data structure for exact searching, and one of the most competitive for proximity searching. Common sense also dictates that an online data structure is less competitive than the full-knowledge, static version. A counter example in proximity searching is the static *Spatial Approximation Tree* (*SAT*), which is slower than its dynamic version (*DSAT*).

In this paper we show that changing only the insertion policy of the *SAT*, leaving every other aspect of the data structure untouched, can produce a systematically faster index. We call the index *Distal Spatial Approximation Tree* (*DiSAT*). We found that even a random insertion policy produce a faster version of the *SAT*, which explains why the *DSAT* is faster than *SAT*. In brief, the *SAT* is improved by selecting distal, instead of proximal, nodes. This is the exact opposite of the insertion policy proposed in the original paper, and can be used in main or secondary memory versions of the index.

We tested our approach with representatives of the state of the art in exact proximity searching. As it happens often in experimental setups, there are no absolute winners in all the aspects tested. Our data structure has no parameters to tune-up and a small memory footprint. In addition it can be constructed quickly. Our approach is among the most competitive, those outperforming *DiSAT* achieve this at the expense of larger memory usage or an impractical construction time.

1. Introduction

Data driven applications often require to identify the objects in a database which are near a given query. For large databases and/or expensive distance computations a sequential scan is prohibitive. Two other examples are *query by content* in an image, audio or video repository, and *quasi-duplicate detection*. This kind of applications cannot use standard classification techniques because the number of classes is unbounded (every object in the collection can be considered a class) and hence a plausible solution is to query the entire collection of objects. In the above setup a sequential solution does not scale well for large instances. If the data fits in main memory, which is the case of modern servers which can easily accommodate 4 Tb of RAM, and the disk is not accessed at all; the operation with lead complexity will be computing distances between objects.

If the number of transactions in a repository is large enough, the cost of building an index can be amortized over a large number of proximity queries. Over the many possible choices of indexes, an index with a small memory footprint is preferred, spending as few bits per database object as possible. This way all the available memory can be used to store the objects avoiding fetching them from secondary memory. This brings the need of optimizing a

* Corresponding author.
E-mail addresses: elchavez@cicese.mx (E. Chávez), vlud@unsl.edu.ar (V. Ludueña), nreyes@unsl.edu.ar (N. Reyes), proggero@unsl.edu.ar (P. Roggero).

http://dx.doi.org/10.1016/j.is.2015.10.014
0306-4379/

data structure for getting so much discriminative power using the same number of bits per node.

In spite of a long standing quest for analyzing the performance of indexing algorithms for proximity searching [40,44], the relatively weak postulates of the metric properties prevents making strong predictions and theoretically modeling the data structures. New indexing methods and data structures are tested against a standard benchmark to establish improvements over previous work. It is customary in the literature to consider only distance computations to compare main memory indexes, the argument is that the total search cost will be driven by the most expensive operation. In this case this operation will be computing distances between objects. This simplification does not take into account the full memory hierarchy, a fine grained analysis of memory usage and cache optimizations are needed for a real world deployment. The computed distances complexity model is, however, easier to compare and in general it will adhere to searching time performance. In this paper we made experiments comparing our approach with the state of the art with both the normalized total search time (which is the speedup over a sequential scan) and the computed distances.

We briefly state the problem in a more formal way to continue the discussion. A metric space is composed by a universe of objects $\mathbb{U}$, and a distance function $d: \mathbb{U} \times \mathbb{U} \to \mathbb{R}$, such that for any $x, y, z \in \mathbb{U}$, $d(x,y) > 0$, $d(x,y) = 0 \iff x = y$, $d(x,y) = d(y,x)$ (symmetry), and obeying the triangle inequality: $d(x,z) + d(z,y) \geq d(x,y)$.

Proximity queries are usually of two types, for a given database $S \subseteq \mathbb{U}$ with size $|S| = n$, $q \in \mathbb{U}$ and $r \in \mathbb{R}^+$, $(q,r)_d = \{x \in S | d(q,x) \leq r\}$ denote a *range query*. The other type of query is the k *nearest neighbor*, denoted $kNN_d(q)$, which retrieves the k closest elements to q in S, formally it retrieves the set $R \subseteq S$ such that $|R| = k$ and $\forall u \in R, v \in S - R$, $d(q,u) \leq d(q,v)$. In our setup, only the distance function can be used in the index design and traversal, as opposed to spatial indexing techniques using coordinate information. An extended catalog of indexes for searching in metric spaces can be found in [46,14,3,53]. Most indexes use the triangle inequality to avoid a sequential scan. The distance between the query and the database objects can be estimated by precomputing some distances to a distinguished subset of the database [31], this briefly conforms the *pivot based* indexes. Other common technique dubbed *local partitioning* group data into spatially compact regions.

Our focus is in *exact proximity searching* as opposed to approximate algorithms where accuracy can be traded off for efficiency. However, there are generic techniques to convert any exact algorithm into approximate by using a form of aggressive pruning, as described for example, in [18].

Range queries are more fundamental than k-nearest neighbor queries, it is enough to use the minimum radius such that the output of the range query have k elements. In [29] the authors propose an algorithm to find the proper query radius without computing additional distances. Due to this optimal result, it is enough to consider range searching in the experimental analysis.

The open challenge in proximity searching is indexing data which is *intrinsically high dimensional*. Even if the objects are not assumed to have coordinates the concept of dimensionality can be translated to metric spaces as well [5,14]. One common characterization of the intrinsic high dimensionality consider only the histogram of distances between database objects. An *easy* instance will have a small mean and large standard deviation, while a *difficult* instance will be the converse; a large mean and a small standard deviation, as described in [12]. The literature in metric access methods is vast, and with different application scenarios. The interested reader is referred to one of the many books and surveys published [14,53,46,28].

The *Spatial Approximation Tree* (*SAT*) is an index based on an alternative approach: rather than dividing the search space, approach the query spatially. Start at a given object in the database and get iteratively closer to the query [37,38,30]. Apart from being algorithmically interesting by itself, it has been shown that the *SAT* gives an attractive trade-off between memory usage, construction time, and search performance.

The *Dynamic Spatial Approximation Tree* (*DSAT*) [39] is an online version of the *SAT*. It is designed to allow dynamic insertions and deletions without increasing the construction cost with respect to the *SAT*. A very surprising and unintended feature of the *DSAT* is the boosting in the searching performance. The *DSAT* is faster in searching even if at construction it has less information than the static version of the index. For the *DSAT* the database is unknown beforehand and the objects arrive to the index at random as well as the queries. A dynamic data structure cannot make strong assumptions about the database and will not have statistics about all the database. Conversely, the *SAT* is a static data structure which, in principle, could take advantage of the *full knowledge* of the database. However in [39] it is shown that *DSAT* is more efficient for searching than the *SAT*. This apparently odd behavior has been puzzling researchers for some time.

We have found the key reason for the performance improvement of the *DSAT* when compared to the *SAT*, this paper is devoted to the discussion of this finding. The central idea is to increase the separation between hyperplanes, which in turn decreases the size of the covering radius. Those two are the discarding rules of *SAT*. The performance improvement consists in selecting distal nodes instead of the proximal nodes selected in the original algorithm. The final result is a very simple tweak lowering the number of distance computations with respect to SAT in every database we tested and for every range of intrinsic dimensions. The *DiSAT*, the new data structure, matches the performance of a competitive index, the *List of Clusters* (*LC*), which is efficient for range queries, specially in high dimensions [35]. The main problem with the *LC* is the construction time and finding the correct parameter (the cluster size), which implies building the index several times. In contrast, our approach is faster to build and with a better tradeoff than the original *SAT*.

This paper sum up preliminary ideas discussed in CCE 2011, Mérida, México [16], where we explored some