Information Systems 60 (2016) 50–68

Contents lists available at ScienceDirect

Information Systems

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

Singleton indexes for nearest neighbor search

E.S. Tellez [a], G. Ruiz [b,1], E. Chavez [c,*,1]

[a] CONACYT Research Fellow / INFOTEC, Mexico
[b] Universidad Michoacana de San Nicolás de Hidalgo, Mexico
[c] CICESE, Mexico

ARTICLE INFO

Article history:
Received 16 June 2015
Received in revised form
20 August 2015
Accepted 3 March 2016
Recommended by: F. Korn
Available online 10 March 2016

Keywords:
Nearest neighbor search
Auto-tuning indexes
Metric indexes
Pivot selection

ABSTRACT

The *nearest neighbor search problem* is fundamental in computer science, and in spite of the effort of a vast number of research groups, the instances allowing an efficient solution are reduced to databases of objects of small intrinsic dimensions. For intrinsically high-dimensional data, the only possible solution is to compromise and use approximate or probabilistic approaches. For the rest of the instances in the middle, there is an overwhelmingly large number of indexes of *claimed* good performance. However, the problem of parameter selection makes them unwieldy for use outside of the research community. Even if the indexes can be tuned correctly, either the number of operations for the index construction and tuning is prohibitively large or there are obscure parameters to tune-up. Those restrictions force users from different fields to use brute force to solve the problem in real world instances.

In this paper, we present a family of indexing algorithms designed for end users. They require as input, the database, a query sample and the amount of space available. Our building blocks are standard discarding rules, and the indexes will add routing objects such as pivots, hyperplane references or cluster centroids. Those indexes are built incrementally and will self-tune by greedily searching for a global optimum in performance.

We experimentally show that using this oblivious strategy our indexes are able to outperform state of the art, manually fine-tuned indexes. For example, our indexes are twice as fast than the fastest alternative (LC, EPT or VPT) for most of our datasets. In the case of LC, the faster alternative for high dimensional datasets, the difference is smaller than 5%. In the same case, our indexes are at least one order of magnitude faster to build. This superior performance is maintained for large, high dimensional datasets (100 million 12-dimensional objects). In this benchmark, our best index is two times faster than the closest alternative (VPT), six times faster than the majority of indexes, and more than sixty times faster than the sequential scan.

1. Introduction

Nearest neighbor search is a pervasive problem in computer science. It appears in many applications such as textual and multimedia information retrieval, machine learning, streaming compression, lossless and lossy compression, bioinformatics, and biometric identification and authentication [9,25], just to name a few.

* Corresponding author.
E-mail addresses: eric.tellez@infotec.mx (E.S. Tellez), gruiz@dep.fie.umich.mx (G. Ruiz), elchavez@cicese.mx (E. Chavez).
[1] Partially funded by (CONACyT Grant 179795) Mexico.

http://dx.doi.org/10.1016/j.is.2016.03.003
0306-4379/

Some applications, e.g. multimedia databases, resort to intermediate representations such as vectors, sets, or strings of symbols. Those representations often produce intrinsically high-dimensional datasets, which in turn may lead to an exhaustive, sequential search at query time even if using an index. This is because nearest neighbor searching is known to be exponentially difficult on the intrinsic dimension of the data as reported in several places (e.g. [9,23]). In those situations, the only plausible solution is to use approximate or probabilistic methods, such that speed is traded for the quality of the solution. Examples of approximate techniques are [26,6,12,1,15,14].

We aim at tractable instances of datasets, where exact solutions are essential. There are applications where an approximate or a probabilistic approach cannot be used. Think for example in biometric identification. In this application, neither a miss (failing to identify the nearest neighbor of an object) nor a false claim (giving an output which is not the nearest neighbor) are acceptable because both lead to a failure of the identification system. For this particular example, the only possible solution is a sequential scan over all the objects in the database. The usual way to scale such a system is by using massive parallelism. Exact proximity searching is also interesting from a pure academic perspective.

It is hard to draw a line between tractable instances and those which only accept an approximate solution due to its high intrinsic dimensionality. One of the sources of this ambiguity is a large number of potential solutions using indexes with claimed low complexity. If a practitioner looks for a solution, the efficiency claims of many papers could be misleading. We will analyze those factors from this practical perspective.

The most sensitive issue is the absence of a complexity model capable of capturing the behavior of an index in realistic circumstances. This limitation implies that indexes would be compared experimentally. Even in this setup there are two alternatives, the first one is to count the number of distance computations as the yardstick for index comparison. The rationale behind this choice is to consider distance computation as the leading cost operation, which in turn should allow comparing different indexes using disparate datasets.

One problem with this approach is that the intrinsic dimensionality of the data is a critical factor in the performance; hence the supposed independence of the dataset vanishes. The other alternative is to use standard benchmarks to compare all the indexes, and using the average time spent on queries as the yardstick. This method has the disadvantage of being unable to compare between indexes belonging to different authors, in different computer systems, and different papers without implementing everything each time. To avoid this disadvantage we normalized the total query time using as reference a sequential scan. While this measure still hides many practical issues, like cache usage, and the workload in a multi-user environment; it will give a better guide for practitioners.

An example of hidden cost using the number of distance computations model is the sequential scan over the data to filter. Clear examples are the AESA algorithm [27] for exact proximity searching, and the Permutation based index [5] in approximate proximity searching. The combination of a relatively cheap distance function and high internal cost can lead to a putative fast index when counting the distance computations, which will be slow in practice.

An additional source of unfairness in the comparison of indexes is the memory usage and the preprocessing time overhead for index construction and maintenance. Some indexes are claimed to be competitive, but the construction cost and/or the space overhead are prohibitive. Below we discuss the most competitive indexes in the literature, along with their possible shortcomings.

1.1. A brief survey of exact indexes

In AESA [27] the index consists of the ($O(n^2)$) distances among the objects in the database stored in a table. For querying, an initial random pivot is selected and using the triangle inequality all the non-relevant objects are filtered. From the remaining objects, the next pivot is selected *close* to the query using some cheaply computed distance. This process is repeated iteratively until only relevant objects remain in the collection. Those remaining objects will be the answer to the query. The claimed complexity of this method is a constant number of distance computations. However, it is necessary to compute a linear number of arithmetic and logical operations, along with a quadratic complexity in preprocessing and storage costs. A restriction of the same idea is presented in LAESA [16], where a constant number of pivots are used; however, the claimed complexity at search time also increases.

Chavez et al. [9] proved that any pivot based metric index requires at least a $O(\log(n))$ random pivots, with n the size of the database. However, the base of the logarithm depends on the intrinsic dimension, needing larger indexes as the intrinsic dimension increases. Above certain intrinsic dimensionality, the optimal number of pivots may not fit in main memory; hence, the rule of thumb is to use as many pivots as they fit. Proceeding in this way reduces the number of distances computed to solve a query, and it is useful for expensive distance functions. However, many of these indexes have a high internal cost, surpassing the cost of a sequential scan. Under this scheme, the selection of pivots is essential to reduce both the memory costs and the number of distances computed. The search time must be several times smaller than the sequential scan, to be of use in practice.

Pivot selection strategies: Since it is critical for the performance of pivot based indexes, a natural question is how to select *good* pivots. A fair rule is to select the pivots randomly, but it is well known that the selection affects the performance of the search. Bustos et al. [2] introduced several pivot-selection strategies. The core of their contribution is a method to compare collections of pivot sets, to decide which one has better performance. The authors claim that a better set of pivots will have a distance distribution of the mapped space with a larger mean value. In particular, they propose an *incremental* selection strategy, which consists in taking a set of N candidate pivots, and select the best one, say p_1. Then, from another set of N