Chapter LXXVII
A Pagination Method for Indexes in Metric Databases

Ana Valeria Villegas
Universidad Nacional de San Luis, Argentina

Carina Mabel Ruano
Universidad Nacional de San Luis, Argentina

Norma Edith Herrera
Universidad Nacional de San Luis, Argentina

INTRODUCTION

Searching for database elements that are close or similar to a given query element is a problem that has a vast number of applications in many branches of computer science, and it is known as proximity search or similarity search. This type of search is a natural extension of the exact searching due to the fact that the databases have included the ability to store new data types such as images, sound, text, and so forth.

Similarity search in nontraditional databases can be formalized using the metric space model (Baeza Yates, 1997; Chavez, Navarro, Baeza-Yates, & Marroquín, 2001; Chávez, & Figueroa, 2004). A metric space is a pair (X, d) where X is a universe of objects and $d: X \times X \rightarrow R^+$ is a distance function that quantifies the similarities among the elements in X. This function d satisfies the properties required to be a distance function:

- $\forall\ x, y \in X,\ d(x,y) \geq 0$ (positiveness)
- $\forall\ x, y \in X,\ d(x,y) = d(y,x)$ (symmetry)
- $\forall\ x, y, z \in X,\ d(x,y) \leq d(x,z) + d(z,y)$ (triangle inequality)

A finite subset $U \subseteq X$, which will be called a database, is the set of objects where the search takes place.

A typical query in a metric database to retrieve similar objects is the range query, denoted by $(q, r)_d$. Given a query $q \in X$ and a tolerance radius r, a range query retrieves all elements from the database that are within a distance r to q; that is $(q, r)_d = \{ x / x \in U \wedge d(x, q) \leq r \}$.

Range queries can be trivially answered by examining the entire database U. Unfortunately, this is generally very costly in real applications. In order to avoid this situation, the database is preprocessed by using an indexing algorithm. An indexing algorithm is an off-line procedure whose

aim is to build a data structure or index designed to save distance computations at query time.

The metric spaces indexing algorithms are based on, first, partitioning the space into equivalence classes and, second, a subsequent indexation of each class. Afterward, at query time, some of these classes can be discarded using the index and an exhaustive search takes place only on the remaining classes. The difference among the existing indexing algorithms is the way they build up the equivalence relation. Basically, two groups can be distinguished: pivot-based algorithms and local partitioning algorithms.

Pivot-Based Algorithms

The idea behind the pivot-based algorithms is as follows. At indexing time, k pivots $\{p_1, p_2,..., p_k\}$ are selected from X and each database element u is mapped to a k-dimensional vector, which represents the respective distances to the pivots, that is, $\phi(u) = (d(u,p_1), d(u,p_2) ,..., d(u,p_k))$. When a range query $(q, r)_d$ is issued, the triangle inequality is used together with the pivots in order to filter out elements in the database, without actually evaluating each distance to the query q. To do this, $\phi(q) = (d(q,p_1), d(q,p_2) ,..., d(q,p_k))$ is computed; if $|d(q,p_i)-d(u,p_i)| > r$ holds for any pivot p_i, then, by the triangle inequality, we know that $d(q,u)>r$ without actually evaluating $d(q,u)$. Only those elements that cannot be discarded using this rule are actually compared against q.

Local Partition Algorithms

In this case, the idea is to divide the database in zones as compact as possible. A set of centers $\{c_1,c_2, ...,c_k\}$ is chosen and each element in the database is associated with its closest center c_i. The set of the elements closer to the center c_i than to any other center forms the class $[c_i]$. There are many possible criteria to discard classes when the index is used to answer a query; the most popular ones are the following:

1. **Hyperplane criterion:** This is the most basic one and the one that best expresses the idea of compact partition. Basically, if c is the center of the class $[q]$ (i.e., the center closest to q), then the query ball does not intersect $[c_i]$ if $d(q,c)+r < d(q,c_i)-r$. In other words, if the query ball does not intersect the hyperplane that divides both its closest center c and c_i, then the ball is totally outside the class of c_i.
2. **Covering radius criterion:** This tries to bound the class $[c_i]$ by considering a ball centered at c_i that contains all the elements of U that lie in that class. The covering radius of c for database U is $cr(c) = max_{u\in[c]\cap U} d(c,u)$. Therefore, we can discard $[c_i]$ if $d(q,c_i)-r > cr(c_i)$.

Regardless of the indexing algorithm used, the total time to evaluate a query (similarity search) can be split as T = *# of distances evaluations* x *complexity (d) + CPU extra time + I/O time*, and we would like to minimize T. Based on the assumption that evaluating d is so costly, most of the research on metric spaces has focused on algorithms to discard elements reducing the number of distance evaluations, paying no attention to I/O cost; one exception is the M tree designed specifically for secondary storage. However, if the index does not fit into main memory, the I/O cost plays an important role and, consequently, the performance criteria will be both the number of distance evaluations and the number of disk page reads (I/O cost).

We begin presenting a brief explanation of indexes on secondary storage. After that, we introduce our proposal in detail and give an application example. Finally, the conclusions are presented.

INDEXES IN SECONDARY STORAGE

The organization of the memory of the most computer systems is based on a hierarchy of memory