Advanced Data Structures

Amalia Duch

CS-UPC

March 5, 2020

Metric Data Structures

Metric data structures are data structures designed for *similarity* or *proximity* searching, i.e., searching for database elements that are *similar* or *close* to a given query element.

Similarity

Similarity is modeled with a distance function that satisfies the triangle inequality and the set of objects together with the distance function is called a *metric space*.

Time complexity

The distance is generally expensive to compute and so the general goal is to reduce the number of distance evaluations.

Methodology

The methods used to solve this problem consist of building a set of equivalence classes, discarding some of them and exhaustively searching the others.

Metric spaces

Let X denote the universe of valid objects and U, a finite subset of it of size $|U| = n$, be the dictionary (i.e. the set of objects where we search; the database).

Distances

Definition

A *distance function* $d : X \times X \to \mathbb{R}$ denotes a *measure of distance* between objects. It has the following properties:

1. Positiveness: $\forall x, y \in X, d(x, y) \geq 0$.
2. Reflexivity: $\forall x \in X, d(x, x) = 0$.
3. Symmetry: $\forall x, y \in X, d(x, y) = d(y, x)$.
4. Strict positiveness: $\forall x, y \in X, x \neq y, d(x, y) > 0$.

Properties 1–4 ensure only a consistent definition of d and can not be used to save comparisons in a proximity query.

Metrics

If d is a *metric*, i.e., it satisfies the triangle inequality:

$$\forall x, y, z \in X, d(x, y) \leq d(x, z) + d(z, y)$$

then, the pair (X, d) is called a metric space and d can be used to save distance calculations in proximity queries.

Proximity queries

There are basically three types:

- Range query $(q, r)_d$: retrieve all the records in U within distance r to q.

$$(q, r)_d = \{u \in U | d(q, u) \leq r\}$$

- Nearest neighbor query $NN(q)$: retrieve from U the closest element to q.

$$NN(q) = \{u \in U | d(u, q) \leq d(v, q) \forall v \in U\}$$

- k-Nearest neighbor query $NN_k(q)$: retrieve the k closest elements to q in U.

$$NN_k(q) = A \subset U, |A| = k, \forall u \in A d(u, q) \leq d(v, q), \forall v \in U$$

Proximity queries

The total time to evaluate a query is:

T = distance evaluations $\times$ complexity of d + extra CPU time + I/O time.

It is assumed that the extra CPU time + the I/O time are negligible besides the complexity of d.

Vector spaces

If the elements of (X, d) are tuples of real numbers then (X, d) is a finite-dimensional vector space.
The most widely used distance function in this case is the family of L_s distances defined as:

$$L_s((x_1, x_2, \ldots, x_k), (y_1, y_2, \ldots, y_k)) = \left(\sum_{i=1}^{k} |x_i - y_i|^s \right)^{1/s}$$

Curse of dimensionality

- Closest point and range search algorithms have an exponential dependency on the dimension of the space [Chazelle 1994]. This is called the *curse of dimensionality*.
- No technique can cope with intrinsic dimension higher than 20. Higher representational dimensions can be handled by dimensionality reduction techniques [Faloustous and Lin 1995],[Cox and Cox 1994],[Hair etal. 1995].

Discrete distance functions: BKT

Burkhard and Keller Trees (BKTs) [Burkhard and Keller 1973].

- An arbitrary element $p \in U$ is selected as the root of the tree.
- For each distance $i > 0$, we define $u_i = \{u \in U, d(u, p) = i\}$ as the set of all the elements at distance i from the root.
- For any non-empty u_i we build recursively a child of p labeled i.

All the elements selected as roots of subtrees are called *pivots*.

Discrete distance functions: BKT

Search:

- For range queries $(q, r)_d$: begin at root p and enter the i-th children if $d(q,p) - r \leq i \leq d(q,p) + r$, and proceed recursively.
- For nearest neighbor queries $NN(q)$: Let $d_{min} = d(p,q)$ and enter the i-th children if $d(q,p) - d_{min} \leq i \leq d(q,p) + d_{min}$, proceed recursively actualizing d_{min} when required.

Discrete distance functions: FQT

Fixed Query Trees (FQTs) [Baeza-Yates etal. 1994].

- BK tree where all the pivots stored in the nodes at the same level are the same (and therefore not necessarily belong to the set stored at the subtree).
- The actual elements are all stored at the leaves.
- Advantages:
  - One can choose the pivots so they do not depend on how the points are distributed
  - Visiting several points in the same level will save distance comparisons (in detriment of the height of the tree).

Discrete distance functions: FHQT

Fixed Height Query Trees (FQTs) [Baeza-Yates etal. 1994].

- FQTs where all the leaves are at the same height.
- Disadvantage: some leaves are deepest than required.

Continuous distance functions: VPT

Vantage Point Tree (VPT) [Uhlman 1991]

- Binary tree built recursively, taking an arbitrary element p as the root and taking the median of the set of all distances to p,
$$M = \text{m}edian\{d(p,u)|\forall u \in U\}$$
.
- The left subtree of p contains all the elements with distance to p les than or equal to M.
- The right subtree of p contains all the elements with distance to p greater than M.

Search: We enter in left subtree if $d(p,q) - r \leq M$ and in right subtree if $d(p,q) + r \geq M$.

Continuous distance functions: MVPT

M-ary Vantage point tree: VPT extended to a M-ary tree by using the M uniform percentiles instead of just the median.

Continuous distance functions: BST

Bi-Sector Tree (BST) [Kalantari and McDonald 1983]. It is a binary tree built as follows:

- At each node two arbitrary centers c_1 and c_2 are selected.
- Left subtree: contains all the elements in U closer to c_1 than to c_2.
- Right subtree: contains all the elements in U closer to c_2 than to c_1.
- For each subtree store its covering radius (the distance between the center and its farthest child).

Search: We enter into each subtree if $d(q, c_i) - r$ is not larger than the covering radius of c_i.

Continuous distance functions: GHT

Generalized Hyperplane Tree (GHT) [Uhlmann 1991]

- Identical in construction to a BST.
- Search algorithm: Enter into the left subtree if $d(q, c_1) - r < d(q, c_2) + r$ and into the right subtree if $d(q, c_2) - r < d(q, c_1) + r$. Observe that there is possibility to enter into both subtrees.

Continuous distance functions: GNAT

Geometric Near-neighbor Access Tree (GNAT) [Brin 1995]

- GHT extended to a m-ary tree keeping the same essential idea.
- At each level, we select m arbitrary centers $c_1, c_2, \ldots, c_m$ and define $U_i = \{u \in U | d(u, c_i) < d(u, c_j), \forall j \neq i\}$. That is, U_i are the elements closer to c_i than to nay other c_j. From the root, m children are recursively built for each U_i set.
- The induced partition is a Voronoi-like partition.
- Search algorithm: Enter into the i-th subtree if $d(q, c_i) - r < d(q, c_j) + r, \forall j \neq i$. Observe that there is possibility to enter into more than one subtree.

Average Complexities of Existing Approaches

DS	Space	Construction	Claimed Query	Extra CPU
BKT	n ptrs	$n \log n$	n^{α}	-
FQT	$nn \log n$ ptrs	$n \log n$	n^{α}	-
FHQT	nnH ptrs	nH	$\log n^{(b)}$	n^{α}
VPT	n ptrs	$n \log n$	$\log n^{(c)}$	-
MVPT	n ptrs	$n \log n$	$\log n^{(c)}$	-
BST	n ptrs	$n \log n$	not analyzed	-
GHT	n ptrs	$n \log n$	not analyzed	-
GNAT	nm^2 dsts	$nm \log_m n$	not analyzed	-

(a) $0 < \alpha < 1$ specific for each DS.

(b) If $h = \log n$.

(c) Only valid when searching with very small radii.

- J. L. Bentley.
  Decomposable searching problems.
  Information Processing Letters, 8(5):133–136, 1979.
- J. L. Bentley and J. H. Friedman.
  Data structures for range searching.
  ACM Computing surveys, 11(4):397–409, 1979.
- J.L. Bentley and J. H. Friedman.
  Data structures for range searching.
  ACM Computing Surveys, 11(4):397–409, 1979.
- W. A. Burkhard.
  Hashing and trie algorithms for partial match retrieval.
  ACM Transactions on Database Systems, 1(2):175–187, 1976.
- L. Devroye.

Branching processes in the analysis of the height of trees.
Acta Informatica, 24:277–298, 1987.

L. Devroye and L. Laforest.
An analysis of random d-dimensional quadtrees.
SIAM Journal on Computing, 19(5):821–832, 1990.

R. A. Finkel and J. L. Bentley.
Quad trees: a data structure for retrieval on composite key.
Acta Informatica, 4(1):1–9, 1974.

Ph. Flajolet, G. Gonnet, C. Puech, and J. M. Robson.
Analytic variations on quad trees.
Algorithmica, 10:473–500, 1993.

Ph. Flajolet and T. Lafforgue.
Search costs in quad trees and singularity perturbation analysis.

Discrete and Computational Geometry, 12(4):151–175, 1993.

- M. Friedman, F. Baskett, and L. J. Shustek.
  An algorithm for finding nearest neighbors.
  IEEE Transactions on Computing, C–24(10):1000–1006, 1975.

- D. E. Knuth.
  The Art of Computer Programming: Sorting and Searching, volume 3.
  Addison–Wesley, 2nd edition, 1998.

- J. Nievergelt, H. Hinterberger, and K. C. Sevcik.
  The grid file: An adaptable symmetric multikey file structure.
  ACM Transactions on Database Systems, 1(9):38–71, 1984.

- M. Regnier.
  Analysis of the grid file algorithms.
  BIT, 25(2):335–357, 1985.
- R. L. Rivest.
  Partial-match retrieval algorithms.
  SIAM Journal on Computing, 5(1):19–50, 1976.
- H. Samet.
  Deletion in two-dimensional quad-trees.
  Communications of the ACM, 23(12):703–710, 1980.
- H. Samet.
  The Design and Analysis of Spatial Data Structures.
  Addison-Wesley, 1990.