Big Data Analytics

Special Topics for Computer Science
CSE 4095-001  CSE 5095-005

April 14

Fei Wang
Associate Professor
Department of Computer Science and Engineering
fei_wang@uconn.edu
Scalability I
K-d Tree

In computer science, a **k-d tree** (short for k-dimensional tree) is a **space-partitioning** data structure for organizing points in a k-dimensional space. k-d trees are a useful data structure for several applications, such as nearest neighbor searches. k-d trees are a special case of **binary space partitioning trees**.

K-d Tree

A 3-dimensional k-d tree. The first split (red) cuts the root cell (white) into two subcells, each of which is then split (green) into two subcells. Finally, each of those four is split (blue) into two subcells. Since there is no more splitting, the final eight are called leaf cells.

K-d Tree

- The k-d tree is a *binary tree* in which every node is a dimensional point.

- Every non-leaf node can be thought of as implicitly generating a splitting hyperplane that divides the space into two parts, known as *half-spaces*.

  - Points to the left of this hyperplane are represented by the left subtree of that node and points right of the hyperplane are represented by the right subtree.

- The hyperplane direction is chosen in the following way:

  Every node in the tree is associated with one of the k-dimensions, with the hyperplane perpendicular to that dimension's axis. So, for example, if for a particular split the "x" axis is chosen, all points in the subtree with a smaller "x" value than the node will appear in the left subtree and all points with larger "x" value will be in the right subtree. In such a case, the hyperplane would be set by the x-value of the point, and its normal would be the unit x-axis.
K-d Tree

```plaintext
function kdtree (list of points pointList, int depth)
{
    // Select axis based on depth so that axis cycles through all valid values
    var int axis := depth mod k;

    // Sort point list and choose median as pivot element
    select median by axis from pointList;

    // Create node and construct subtrees
    var tree_node node;
    node.location := median;
    node.leftChild := kdtree(points in pointList before median, depth+1);
    node.rightChild := kdtree(points in pointList after median, depth+1);
    return node;
}
```
K-d Tree

Building a static k-d tree from n points has the following worst-case complexity:

- $O(n \log_2 n)$ if an $O(n \log n)$ sort such as Heapsort or Mergesort is used to find the median at each level of the nascent tree;

- $O(n \log n)$ if an $O(n)$ median of medians algorithm is used to select the median at each level of the nascent tree;

- $O(kn \log n)$ if n points are presorted in each of k dimensions using an $O(n \log n)$ sort such as Heapsort or Mergesort prior to building the k-d tree.

- Inserting a new point into a balanced k-d tree takes $O(\log n)$ time.

- Removing a point from a balanced k-d tree takes $O(\log n)$ time.

- Querying an axis-parallel range in a balanced k-d tree takes $O(n^{1-1/k} + m)$ time, where $m$ is the number of the reported points, and $k$ the dimension of the k-d tree.

- Finding 1 nearest neighbour in a balanced k-d tree with randomly distributed points takes $O(\log n)$ time on average.
K-d Tree

k-d trees are not suitable for efficiently finding the nearest neighbour in *high-dimensional* spaces. As a general rule, if the dimensionality is k, the number of points in the data, N, should be $N \gg 2^k$. Otherwise, when k-d trees are used with high-dimensional data, most of the points in the tree will be evaluated and the efficiency is no better than exhaustive search, and approximate nearest-neighbour methods should be used instead.
Ball Tree

In computer science, a ball tree, balltree or **metric tree**, is a **space partitioning** data structure for organizing points in a multi-dimensional space. The ball tree gets its name from the fact that it partitions data points into a nested set of **hyperspheres** known as "balls". The resulting data structure has characteristics that make it useful for a number of applications, most notably nearest neighbor search.
Ball Tree

- A ball tree is a **binary tree** in which every node defines a D-dimensional **hypersphere**, or ball, containing a subset of the points to be searched.

- Each internal node of the tree partitions the data points into two **disjoint** sets which are associated with different balls.

- While the balls themselves may intersect, each point is assigned to one or the other ball in the partition according to its distance from the ball's center.

- Each leaf node in the tree defines a ball and all enumerates all data points inside that ball.
function construct_balltree is
    input:
        D, an array of data points
    output:
        B, the root of a constructed ball tree
    if a single point remains then
        create a leaf B containing the single point in D
        return B
    else
        let c be the dimension of greatest spread
        let L, R be the sets of points lying to the left and right of the median along dimension c
        create B with two children:
            B.pivot = c
            B.child1 = construct_balltree(L),
            B.child2 = construct_balltree(R)
        return B
    end if
end function
Ball Tree