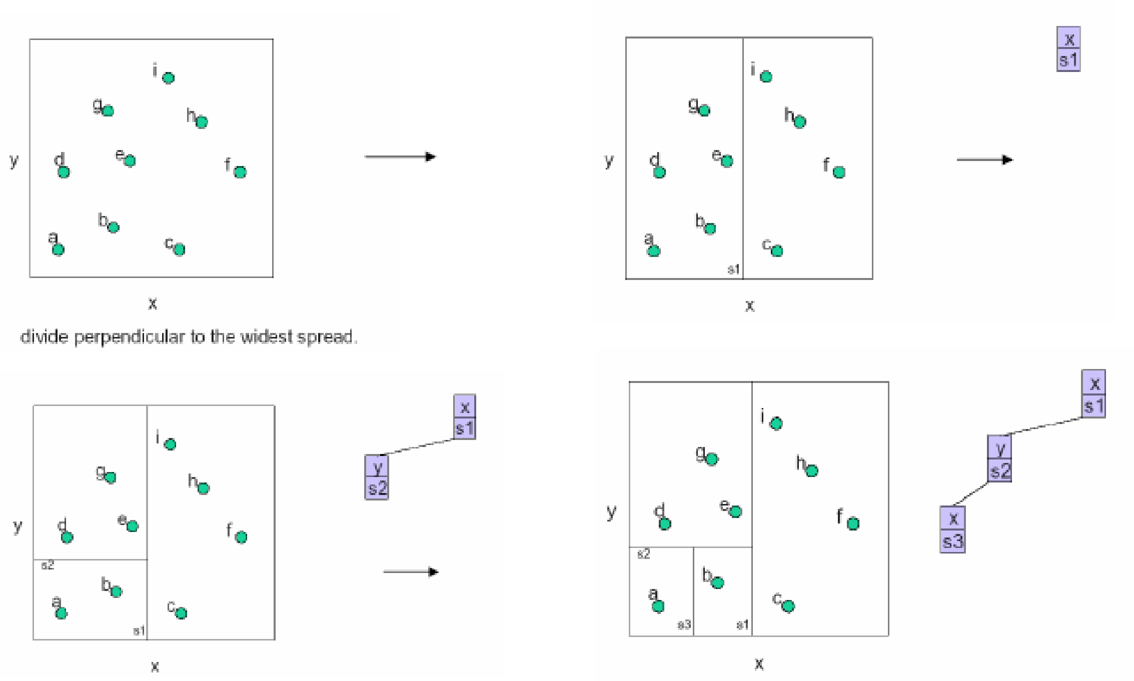
Parallel Machine Learning

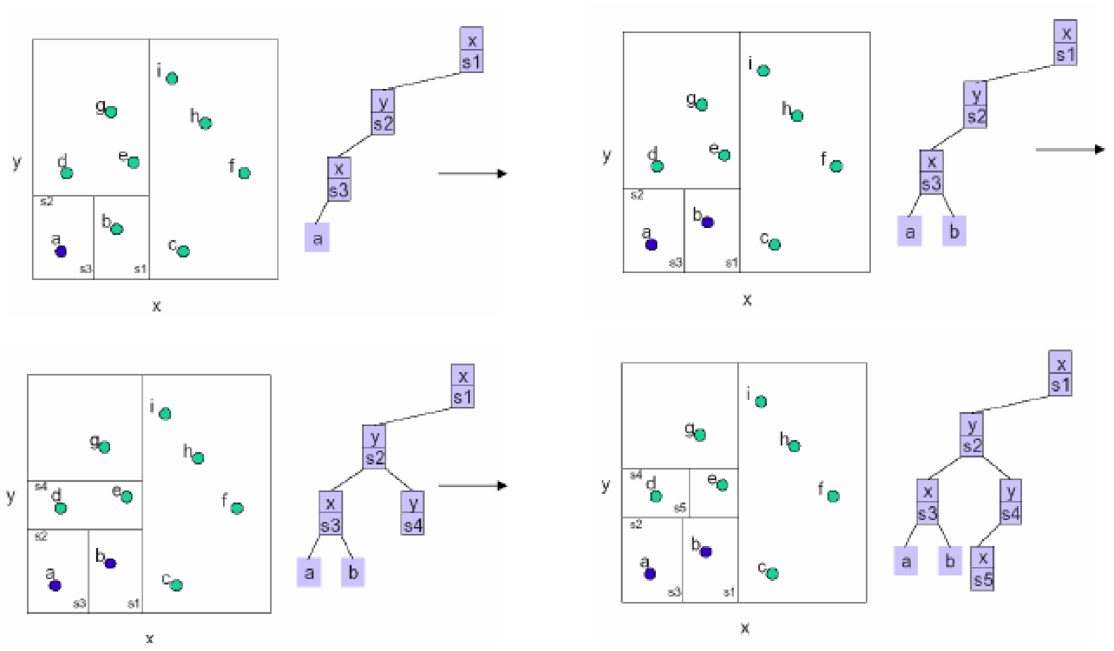
- k-NN used for classification and regression problems
- Commonly used data structure: k-d trees
- For the given multi-dimensional data, construct a k-d tree
- Similar to

k-d Tree Construction

k-d tree construction example

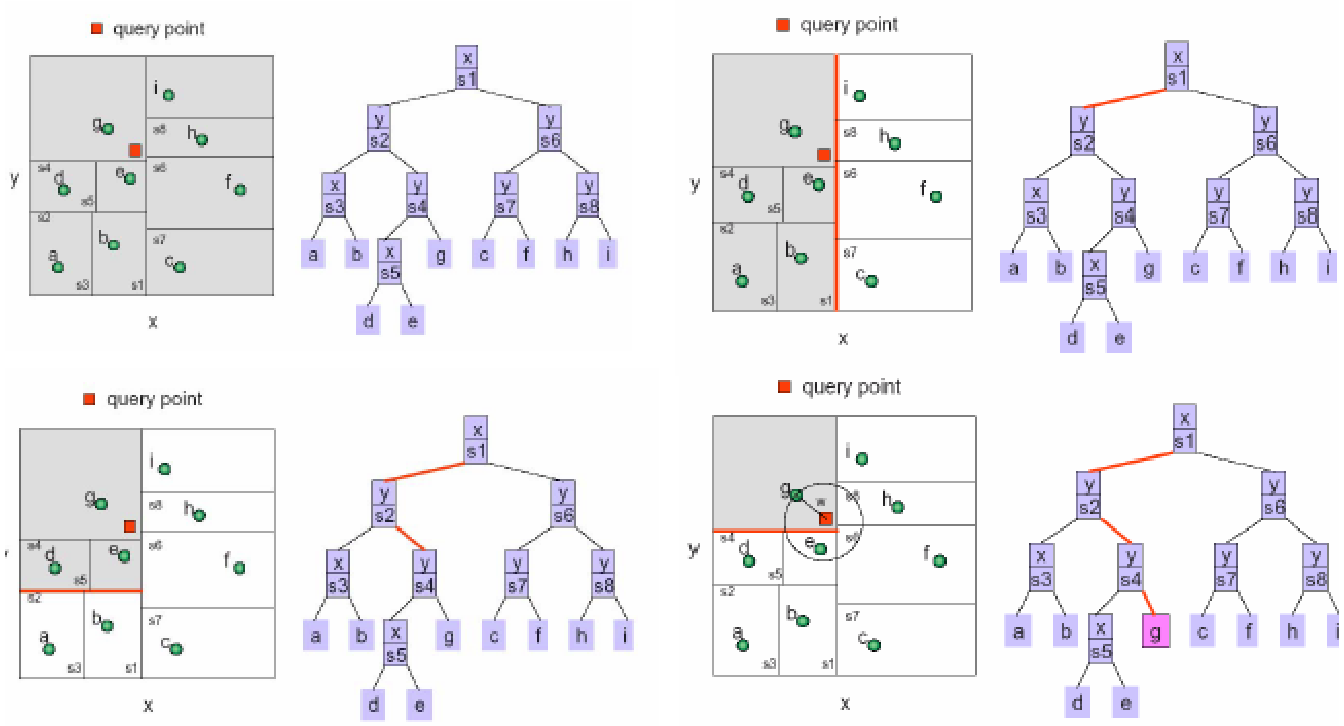


k-d tree construction example



Nearest neighbour search using kd- tree

k-d Tree Nearest Neighbor Search



Two steps in k-NN

1. k-d tree construction
2. K-d tree search

Parallelization

- Option 1:
  - Tree construction: Partition data sets among processors; Each processor constructs local k-d tree
  - Search: Query sent to all processors which perform search in local k-d trees; each processor returns the top neighbors from which the k nearest are chosen

Parallelization

- Option 1: Poor work efficiency, i.e., wasted work

- Option 2:
  - Global k-d tree construction in all the processors
  - For each processors, one half of data is given to one half of processors, and the other half of data given to other processors
  - After this recursive division, the top part of the tree is replicated in all the processors
  - The processors then construct local k-d trees for their subdomains

Searching the global k-d tree

- Query sent to the processors that takes care of the subdomain of the query
- The processor forms the local k nearest neighbors
- Forms a radius based on these neighbors
- Sends the query to the nearby processors consisting of subdomains that are spanned by the radius
- The processors search for points within the radius and send their results to the origin processor

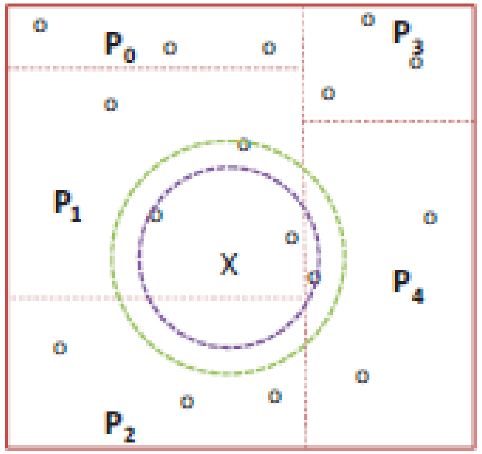


Figure 3. Figure shows the data points (denoted by o) in 2D space divided among 5 nodes. Query point is shown as X. KNN with $k = 3$ is run in node P_1 (owner of X). This returns 3 points owned by P_1 and a max distance (denoted by green circle around X). Only P_2 and P_4 might own points within this radius. KNN is run on P_2 and P_4 for X and the closest 3 points are chosen (within purple circle around X).

Optimization

- Queries can be batched
- Software pipelining can be performed between the stages

Algorithm

Algorithm 1 Finding k-nearest neighbors from the local kd-tree. Input: kd-tree T, Query q, k, search radius, r (default $r = \infty$). Output: A set, R of k nearest neighbors within r.

procedure FINDKNN(T, q, k, r)
    $r' \leftarrow r$; push $(root, 0)$ into S
    while S is not empty **do**
        $(node, d) \leftarrow$ pop from S
        if $node$ is leaf **then**
            for each particle x in $node$ **do**
                compute distance, $d[x]$ of x from q
                if $d[x] < r'$ **then**
                    if $|H| < k$ **then**
                        add x into H
                        if $|H| = k$ **then**
                            $r' \leftarrow H.maxi\_dis$
                    else if $d[x] <$ max distance in H **then**
                        replace the topmost point H by x
                        $r' \leftarrow d[x]$
        else

Algorithm

else
    if $d < r'$ **then**
        $d' \leftarrow q[node.dim] - node.median$
        $d' \leftarrow \sqrt{d * d + d' * d'}$
        $C_1 \leftarrow$ closer child of $node$ from q
        $C_2 \leftarrow$ other child of $node$
        if $d' < r'$ **then**
            push (C_2, d') into S
        push (C_1, d) into S
$R \leftarrow H$

References

- PANDA: Extreme Scale Parallel K-Nearest Neighbor on Distributed Architectures. IPDPS 2016.