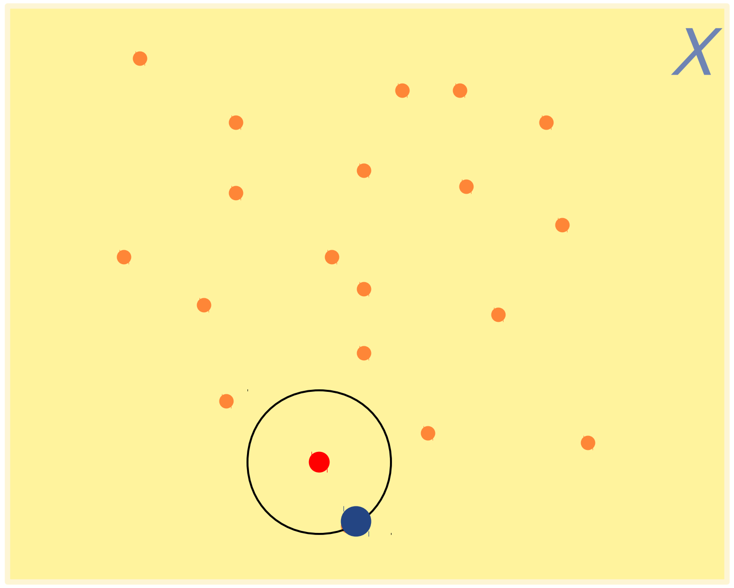
Balanced Box-Decomposition trees for Approximate nearest-neighbor

Manos Thanos (MPLA)
Ioannis Emiris (Dept Informatics)
Computational Geometry

Nearest Neighbor

A set *S* of *n* points is given in some metric space *X*.
Problem: given any query point $q \in X$,
find the point of *S* nearest to *q* *(nearest neighbor, NN).*

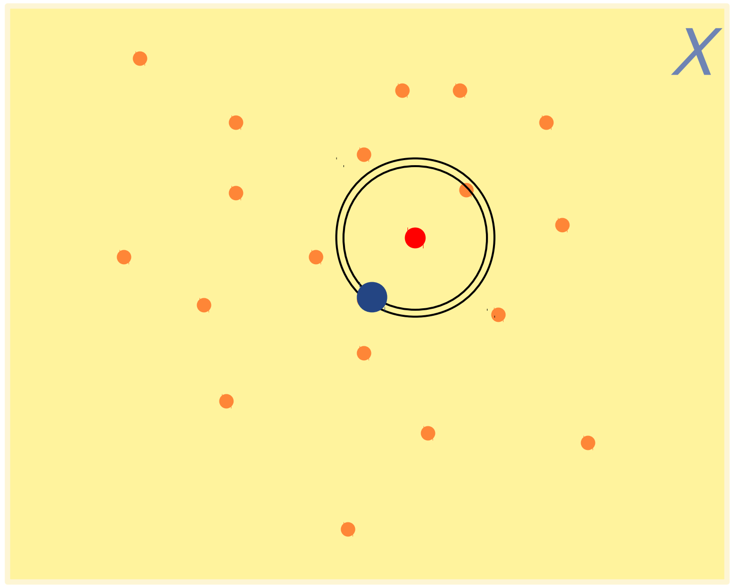
Idea: preprocess S so that, given *q*, the NN is reported quickly.
Efficient exact solutions known for only very limited cases;
hence **approximation algorithms.**



Approximate nearest neighbor

Given a query point q, let $NN(q)$ be the distance from q to its nearest neighbor in S.

For a real parameter $\varepsilon > 0$, point $p \in S$ is an ε-nearest neighbor (ε-NN) of q if the distance from p to q is at most $(1 + \varepsilon)NN(q)$.



NN Formulations

Traditional computational geometry: the space is *Rd*, for **constant *d*,** under the Euclidean norm; ***ε* is an asymptotic quantity** of secondary importance to *n*.

Others treat ***d*** **as an asymptotic quantity** and seek solutions having **no exponential dependence on *d*.**

Others assume the **metric space** possesses a **growth-limiting property**, e.g. constant doubling dimension: twice the ball is included in constant number of balls (true for Euclidean).

Balanced Box-Decomposition tree

[Arya, Mount et al, J.ACM'98]

Each node of the BBD-tree is associated with a cell: this is a *d*-dim rectangle (box), or with the set theoretic **difference of two** such rectangles (boxes), one enclosed in the other.
Each cell is defined by an outer box and an optional inner box.

Each cell will be associated with the set of data points lying within the cell. Points which lie on the boundary between cells may be assigned to either cell.

The **aspect ratio** (ratio of length of longest to shortest side) of the boxes **is bounded by a constant.**

BBD Construction

The BBD-tree is constructed by applying two operations, as long as cell contains >1 points:

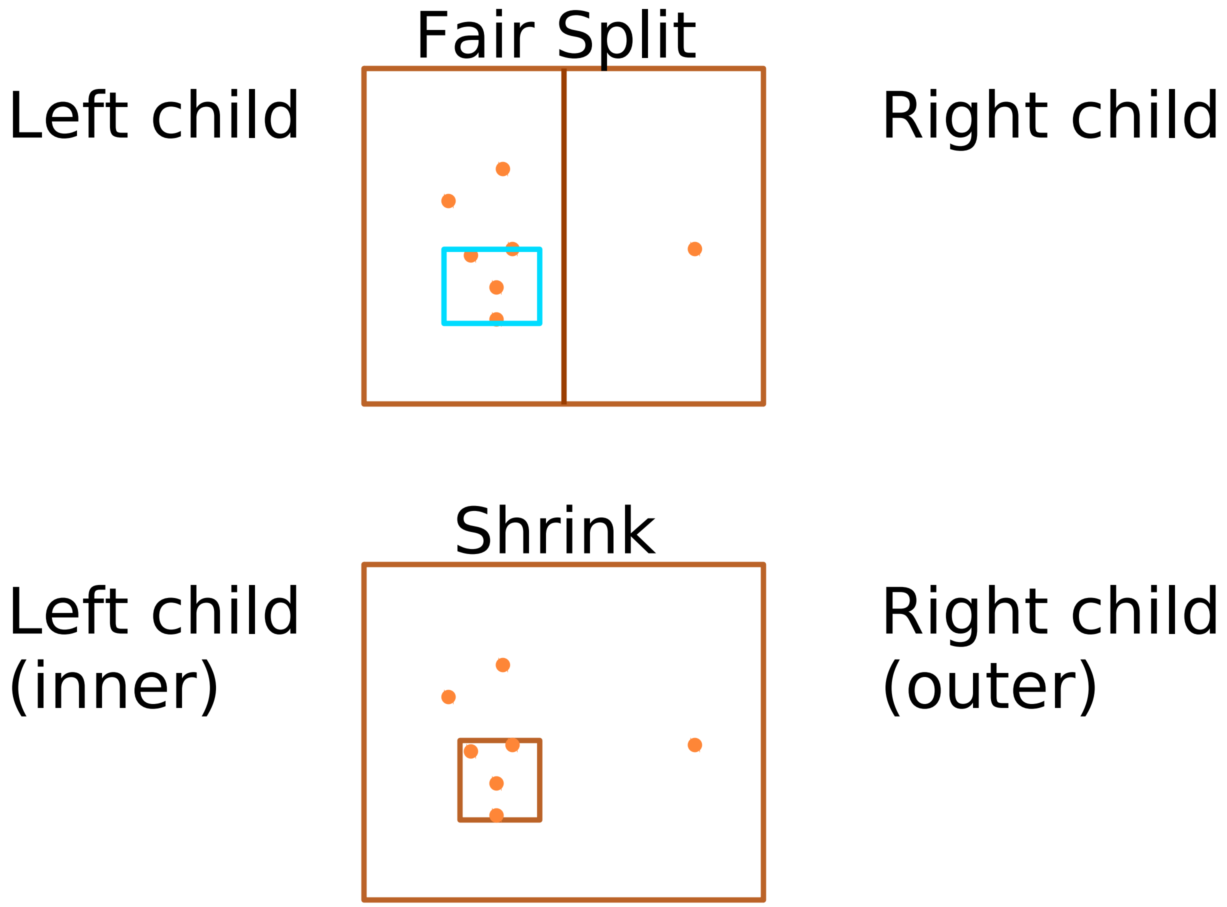
- **(Fair) Splits**
  by a hyperplane parallel to a coordinate plane
  (if inner box exists, do not intersect it)
  Guarantees geometric decrease of children cells

- **Shrinks**
  partitions box into inner and outer
  (if inner box, it lies inside new inner box)
  Guarantees decrease of number of points in children

Two strategies:

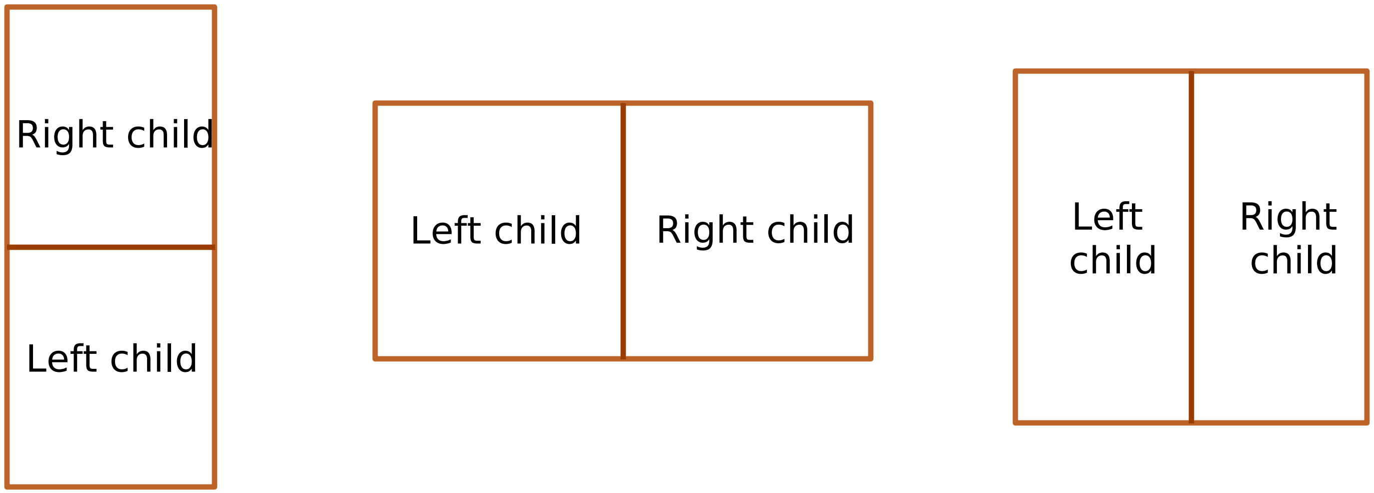
- Splits and Shrinks alternate.
- Always Split unless Shrink is needed

Operations


Fair Split
Left child
Right child
Shrink
Left child (inner)
Right child (outer)

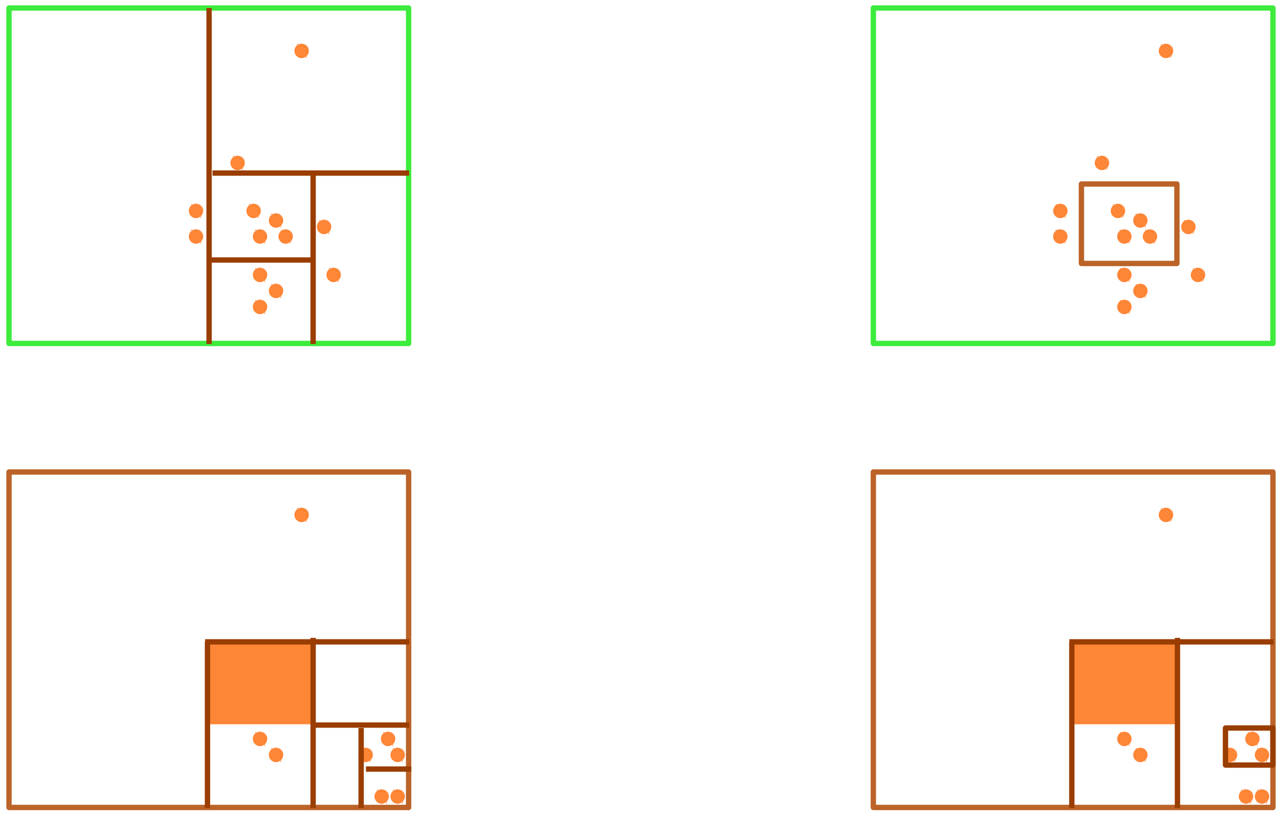
Midpoint splitting

The **midpoint** is used to perform the splitting and shrinking rules.

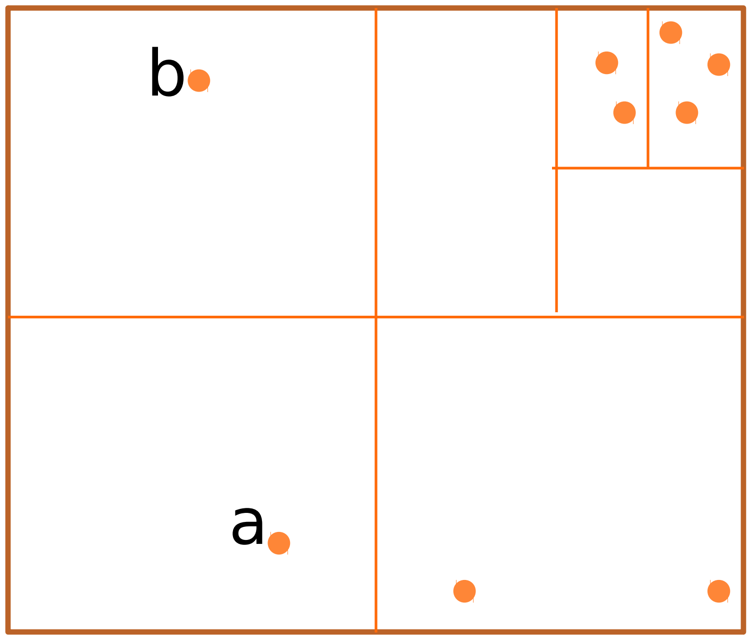


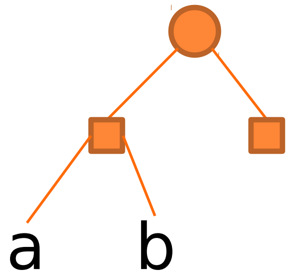
Centroid shrink

Midpoint splits until both children have less than 2/3 of the parent's points

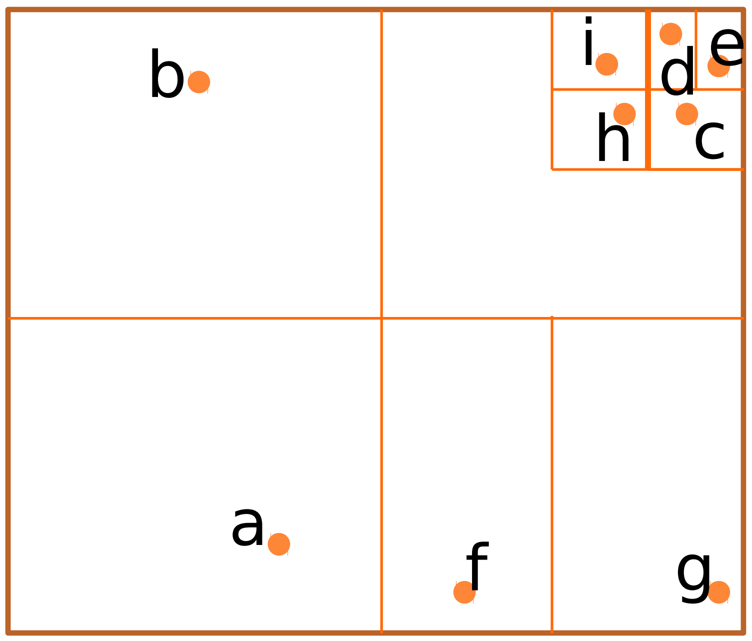


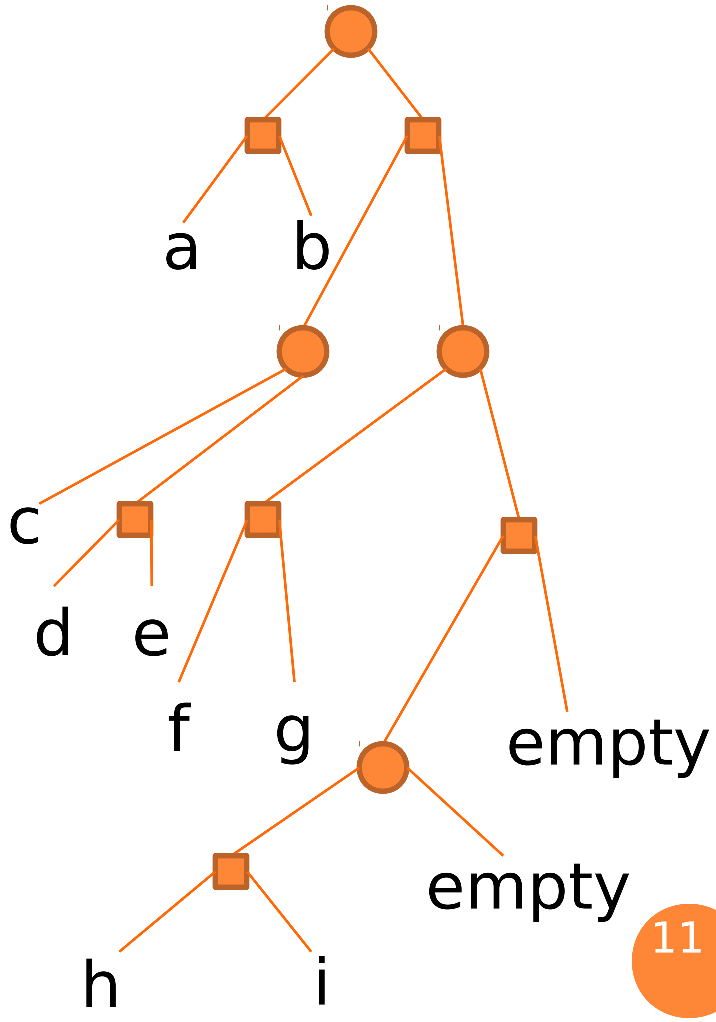
Example I





Example II





Invariants

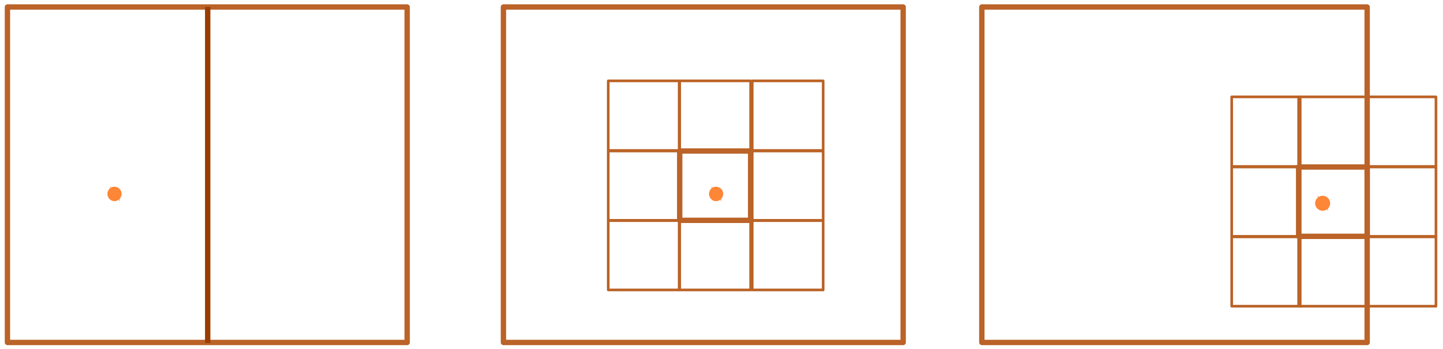
Both operations are performed so that the following **invariants** hold:

All boxes satisfy the **aspect** ratio bound.

If the parent has an inner box, then this box lies entirely within one of the two children. If the operation is a shrink, then this inner box lies within the left (inner) child of the shrink:
Inner boxes are sticky for their enclosing outer boxes.

Stickiness

Important concept that restricts the nature of inner boxes.



BBD Complexity

The cells created by the midpoint rule satisfy the aspect ratio bound and are sticky to their parents.

Splits and shrinks are applied alternately for the construction of the BBD-tree.

The height of the tree is O(*log* n).
In general, with every 4 levels of descent in the tree, the number of points associated with the nodes decreases by at least a factor of 2/3.

Construction in O(*d* *n* log*n*) time.

Search

Each leaf contains up to some constant number of points, e.g. 1, hence *$O(n)$* total space.

Given a point q in R^d, a leaf **containing *q*** is determined in ***O(d* log*n)* time** (point location).

The distance between q and a cell = closest distance between q and any point/part of the cell.
Given q, all cells can be enumerated in order of increasing distance from q. The ***m* nearest** cells can be enumerated in ***O(md* log*n)*** time.

Approximate Nearest Neighbor

Algorithm

- Find the leaf that contains query point *q*.
- Find closest cell to *q*, compute min-distance δ between *q* and the points in that cell.
- While the distance of the next closest cell <δ(1+ε), compute min-distance between *q* and the points in this cell. If this distance <δ, update δ to be this distance.

The *m* closest cells are computed in $O(md \log n)$ time.
Number of cells visited is $c < (1+6d/\varepsilon)^d$.
Thus, the algorithm answers ANN queries in $O(cd \log n)$ time.

ANN software

The implementation of the algorithm is “**ANN**”.

Empirical runtimes on most distributions suggest there is **little or no significant practical advantage** to using the BBD-tree over the kd-tree, enhanced with some improvements (allowing approximation errors, incremental distance calculations).

Results show that,
for even very large ε, the average error is typically at least an order of magnitude **smaller**,
number of cells visited is significantly **smaller** than the huge predicted values $(1+6d/\varepsilon)^d$.

ANN improvement

Improvements were proposed, the best offering:
query time $= O(\log n + 1/\varepsilon^{d-1})$, $O(n)$ space.
These structures are optimal with respect to space, the ε-dependencies in query time are far from optimal.

Space-time tradeoff for ε-NN: Approximate Voronoi Diagrams achieve time $O(\log(n/\varepsilon))$ and space $O((n/\varepsilon^{d-1}) \log(1/\varepsilon))$.
Let $M(n)$ and $T(n)$ denote space and query time.
Then (ignoring log factors) $\boldsymbol{M(n)T^2(n) = O(n/\varepsilon^{d-1})}$.