

Introduction ○○○○	Euclidean Spaces ○○○○○○○○○○	Metric Spaces ○○○○○○○○	Almost Done ○○○
----------------------	--------------------------------	---------------------------	--------------------

On Proximity Searching in Euclidean and Other Metric Spaces

David Mount

University of Maryland, College Park

NASA-GSFC, Nov 2009

Geometric Proximity Search

Geometric Retrieval

Given a set of geometric objects in some space, **store** these objects in a data structure so that **queries** about these objects can be answered **efficiently**.

Proximity Searching

Report (or count) the objects that are **close** to some point/region.

- **Nearest Neighbors**: Find the closest object to the query.
- **Range Searching**: Report the objects within a query shape.

- **Efficiency:** What is the **query time** as a function of the number of objects and the amount of **space** used by the data structure.
- **Space-Time Tradeoffs:** As space **increases**, how fast does query time **decrease**?
- **Computational Complexity:** What are the **best possible** tradeoffs?
- **Approximation:** If small **errors** are allowed, can we answer queries **faster**?

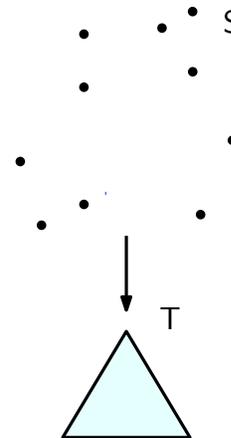
Nearest Neighbor Searching

Nearest Neighbor Searching

Given a set S of n points, preprocess S into a data structure T so that given a query point q , the closest point $p^* \in S$ to q can be reported efficiently.

Assumptions and Goals

- “Intrinsic dimension” is a constant. Ignore factors depending on dimension.
- Ideal space: $O(n)$
- Ideal query time: $O(\log n)$



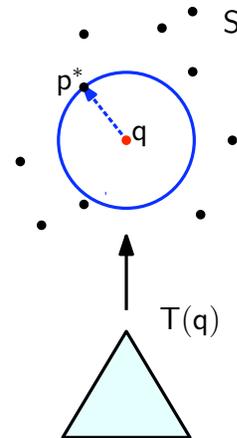
Nearest Neighbor Searching

Nearest Neighbor Searching

Given a set S of n points, preprocess S into a data structure T so that given a query point q , the closest point $p^* \in S$ to q can be reported efficiently.

Assumptions and Goals

- “Intrinsic dimension” is a constant. Ignore factors depending on dimension.
- Ideal space: $O(n)$
- Ideal query time: $O(\log n)$



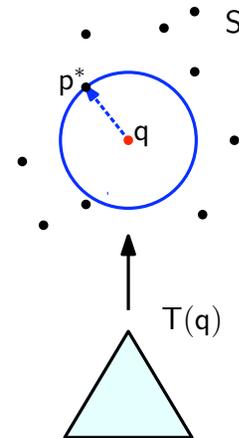
Nearest Neighbor Searching

Nearest Neighbor Searching

Given a set S of n points, preprocess S into a data structure T so that given a query point q , the closest point $p^* \in S$ to q can be reported efficiently.

Assumptions and Goals

- “Intrinsic dimension” is a constant. Ignore factors depending on dimension.
- Ideal space: $O(n)$
- Ideal query time: $O(\log n)$



Distance

Metric Space

... is a set S and function $d : S \times S \Rightarrow R$, for any $x, y, z \in S$:

- **Non-negativity:** $d(x, y) \geq 0$ and $d(x, y) = 0$ iff $x = y$.
- **Symmetry:** $d(x, y) = d(y, x)$.
- **Triangle inequality:** $d(x, z) \leq d(x, y) + d(y, z)$.

Euclidean distance: $S = \mathbb{R}^2$, $d(\mathbf{x}, \mathbf{y}) = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2}$.

Edit distance: Number of edit ops to map one string to another.

Geodesic distance: Length of the shortest path on a manifold.

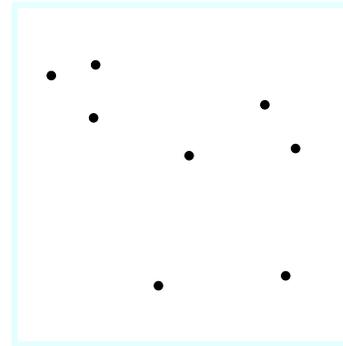
- Nearest neighbor searching in **Euclidean spaces**:
 - Voronoi diagrams
 - Approximate NN-searching and BBD trees
 - Approximate Voronoi diagrams (AVDs)
- Nearest neighbor searching in **doubling spaces**:
 - r -nets and net trees
 - Doubling oracle and the weakly explicit model
 - AVDs in metric spaces

Euclidean Nearest Neighbors in the Plane

Voronoi Diagram

Given a point set $S \subset \mathbb{R}^2$ the **Voronoi diagram** is a subdivision of the plane into regions according to which point of S is closest.

- **Preprocessing:** Build the Voronoi diagram and a point location structure for S .
 $\Rightarrow O(n \log n)$ time, $O(n)$ space.
- **Query Processing:** Locate the cell containing q . $\Rightarrow O(\log n)$ time.

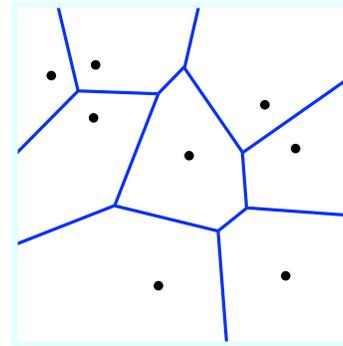


Euclidean Nearest Neighbors in the Plane

Voronoi Diagram

Given a point set $S \subset \mathbb{R}^2$ the **Voronoi diagram** is a subdivision of the plane into regions according to which point of S is closest.

- **Preprocessing:** Build the Voronoi diagram and a point location structure for S .
 $\Rightarrow O(n \log n)$ time, $O(n)$ space.
- **Query Processing:** Locate the cell containing q . $\Rightarrow O(\log n)$ time.

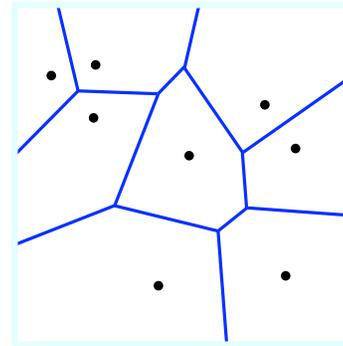


Euclidean Nearest Neighbors in the Plane

Voronoi Diagram

Given a point set $S \subset \mathbb{R}^2$ the **Voronoi diagram** is a subdivision of the plane into regions according to which point of S is closest.

- **Preprocessing:** Build the Voronoi diagram and a point location structure for S .
 $\Rightarrow O(n \log n)$ time, $O(n)$ space.
- **Query Processing:** Locate the cell containing q . $\Rightarrow O(\log n)$ time.

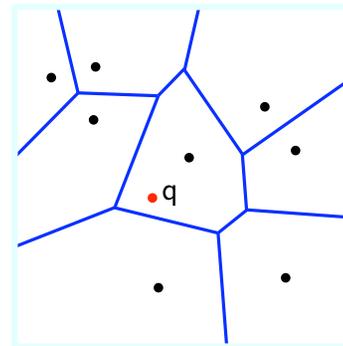


Euclidean Nearest Neighbors in the Plane

Voronoi Diagram

Given a point set $S \subset \mathbb{R}^2$ the **Voronoi diagram** is a subdivision of the plane into regions according to which point of S is closest.

- **Preprocessing:** Build the Voronoi diagram and a point location structure for S .
 $\Rightarrow O(n \log n)$ time, $O(n)$ space.
- **Query Processing:** Locate the cell containing q . $\Rightarrow O(\log n)$ time.

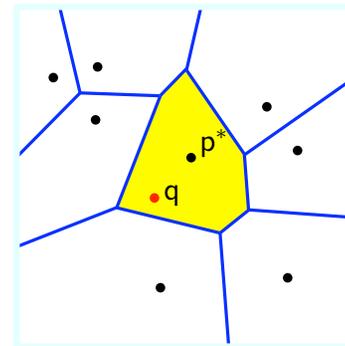


Euclidean Nearest Neighbors in the Plane

Voronoi Diagram

Given a point set $S \subset \mathbb{R}^2$ the **Voronoi diagram** is a subdivision of the plane into regions according to which point of S is closest.

- **Preprocessing:** Build the Voronoi diagram and a point location structure for S .
 $\Rightarrow O(n \log n)$ time, $O(n)$ space.
- **Query Processing:** Locate the cell containing q . $\Rightarrow O(\log n)$ time.



- ### Problems in Higher Dimensions
- The complexity of the Voronoi diagram grows **rapidly** — $O(n^{\lceil d/2 \rceil})$.
 - Planar point location methods **do not** extend to higher dimensions.

Approximate Nearest Neighbor Searching

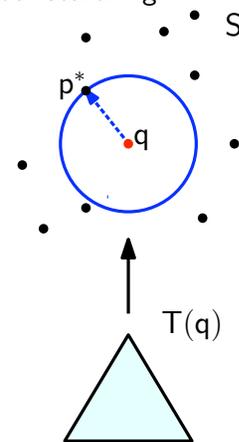
Motivates interest in **approximate** nearest neighbor searching.

Approximate Nearest Neighbor Searching

Given q and $\varepsilon > 0$, return $p \in S$ so that

$$d(q, p) \leq (1 + \varepsilon)d(q, p^*),$$

where $p^* \in S$ is the nearest neighbor of q . We call p an **ε -approximate nearest neighbor** of q .



Approximate Nearest Neighbor Searching

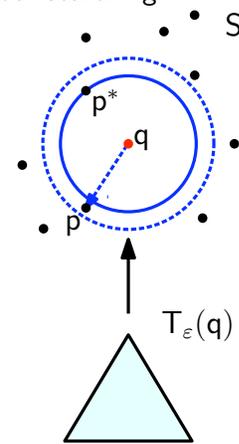
Motivates interest in **approximate** nearest neighbor searching.

Approximate Nearest Neighbor Searching

Given q and $\varepsilon > 0$, return $p \in S$ so that

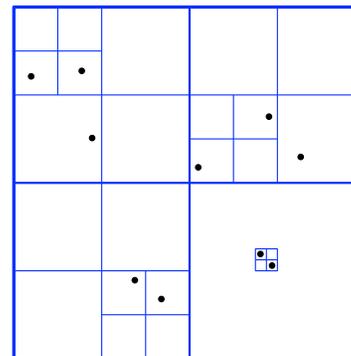
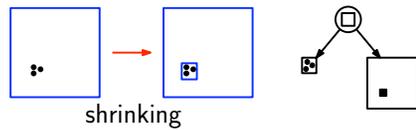
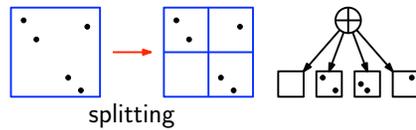
$$d(q, p) \leq (1 + \varepsilon)d(q, p^*),$$

where $p^* \in S$ is the nearest neighbor of q . We call p an **ε -approximate nearest neighbor** of q .



Box-Decomposition Tree: A Better Quadtree

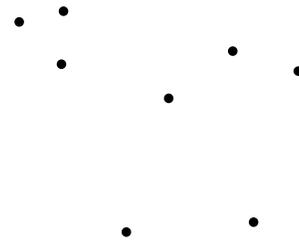
- **Cell:** Difference of two quadtree boxes (inner and outer).
- **Centroid Decomposition:** Used to guarantee $O(\log n)$ depth.



Nearest Neighbor Searching with BBD trees

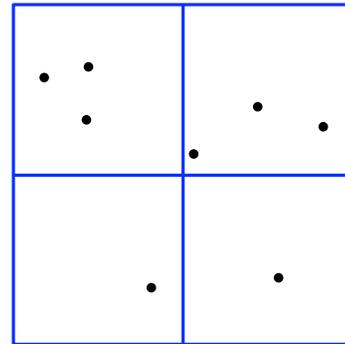
ϵ -NN Searching with BBD trees

- **Preprocessing:** $O(n)$ space.
- **Query Processing:**
 - Locate the cell containing q ($O(\log n)$ time).
 - Establish initial search radius.
 - Recursively visit nodes only if they are close enough to offer a closer point.
- **Query time:** $O(\log n + (1/\epsilon)^d)$.



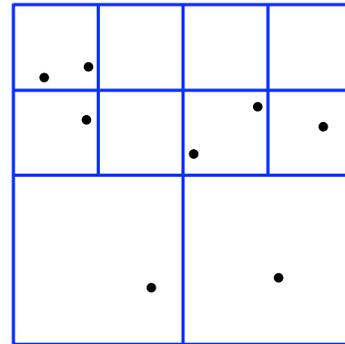
Nearest Neighbor Searching with BBD trees

- ϵ -NN Searching with BBD trees
- **Preprocessing:** $O(n)$ space.
 - **Query Processing:**
 - Locate the cell containing q ($O(\log n)$ time).
 - Establish initial search radius.
 - Recursively visit nodes only if they are close enough to offer a closer point.
 - **Query time:** $O(\log n + (1/\epsilon)^d)$.



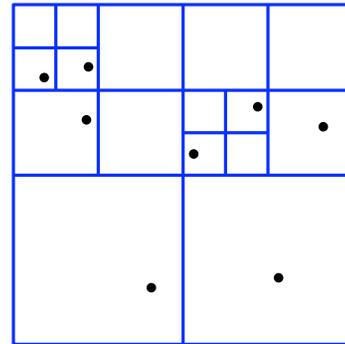
Nearest Neighbor Searching with BBD trees

- ϵ -NN Searching with BBD trees
- **Preprocessing:** $O(n)$ space.
 - **Query Processing:**
 - Locate the cell containing q ($O(\log n)$ time).
 - Establish initial search radius.
 - Recursively visit nodes only if they are close enough to offer a closer point.
 - **Query time:** $O(\log n + (1/\epsilon)^d)$.



Nearest Neighbor Searching with BBD trees

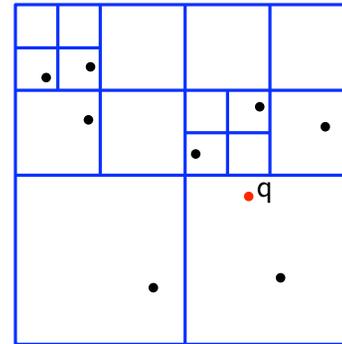
- ϵ -NN Searching with BBD trees
- **Preprocessing:** $O(n)$ space.
 - **Query Processing:**
 - Locate the cell containing q ($O(\log n)$ time).
 - Establish initial search radius.
 - Recursively visit nodes only if they are close enough to offer a closer point.
 - **Query time:** $O(\log n + (1/\epsilon)^d)$.



Nearest Neighbor Searching with BBD trees

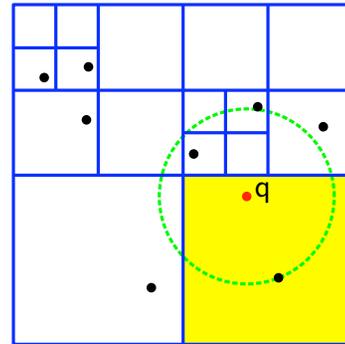
ϵ -NN Searching with BBD trees

- **Preprocessing:** $O(n)$ space.
- **Query Processing:**
 - Locate the cell containing q ($O(\log n)$ time).
 - Establish initial search radius.
 - Recursively visit nodes only if they are close enough to offer a closer point.
- **Query time:** $O(\log n + (1/\epsilon)^d)$.



Nearest Neighbor Searching with BBD trees

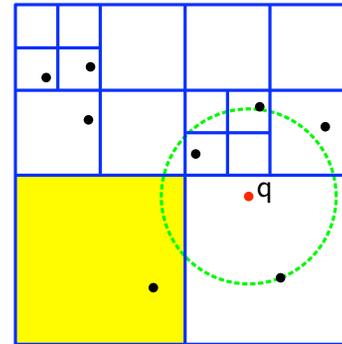
- ϵ -NN Searching with BBD trees
- **Preprocessing:** $O(n)$ space.
 - **Query Processing:**
 - Locate the cell containing q ($O(\log n)$ time).
 - Establish initial search radius.
 - Recursively visit nodes only if they are close enough to offer a closer point.
 - **Query time:** $O(\log n + (1/\epsilon)^d)$.



Nearest Neighbor Searching with BBD trees

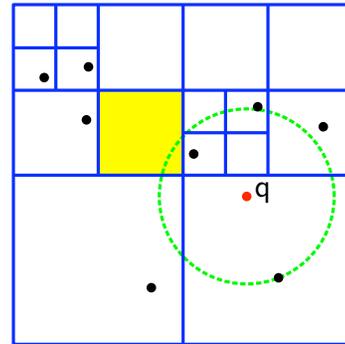
ϵ -NN Searching with BBD trees

- **Preprocessing:** $O(n)$ space.
- **Query Processing:**
 - Locate the cell containing q ($O(\log n)$ time).
 - Establish initial search radius.
 - Recursively visit nodes only if they are close enough to offer a closer point.
- **Query time:** $O(\log n + (1/\epsilon)^d)$.



Nearest Neighbor Searching with BBD trees

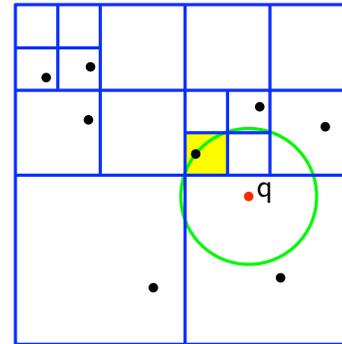
- ϵ -NN Searching with BBD trees
- **Preprocessing:** $O(n)$ space.
 - **Query Processing:**
 - Locate the cell containing q ($O(\log n)$ time).
 - Establish initial search radius.
 - Recursively visit nodes only if they are close enough to offer a closer point.
 - **Query time:** $O(\log n + (1/\epsilon)^d)$.



Nearest Neighbor Searching with BBD trees

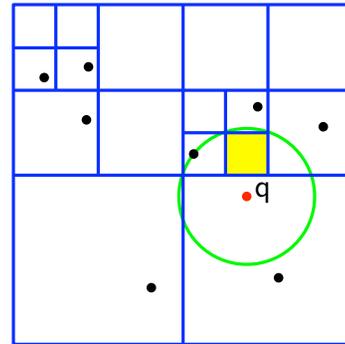
ϵ -NN Searching with BBD trees

- **Preprocessing:** $O(n)$ space.
- **Query Processing:**
 - Locate the cell containing q ($O(\log n)$ time).
 - Establish initial search radius.
 - Recursively visit nodes only if they are close enough to offer a closer point.
- **Query time:** $O(\log n + (1/\epsilon)^d)$.



Nearest Neighbor Searching with BBD trees

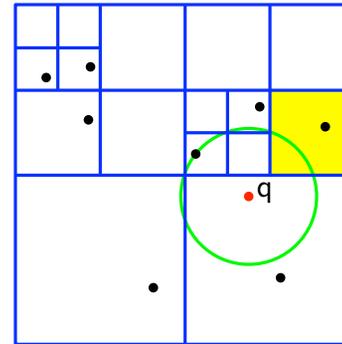
- ϵ -NN Searching with BBD trees
- **Preprocessing:** $O(n)$ space.
 - **Query Processing:**
 - Locate the cell containing q ($O(\log n)$ time).
 - Establish initial search radius.
 - Recursively visit nodes only if they are close enough to offer a closer point.
 - **Query time:** $O(\log n + (1/\epsilon)^d)$.



Nearest Neighbor Searching with BBD trees

ϵ -NN Searching with BBD trees

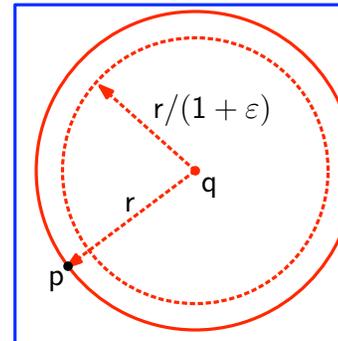
- **Preprocessing:** $O(n)$ space.
- **Query Processing:**
 - Locate the cell containing q ($O(\log n)$ time).
 - Establish initial search radius.
 - Recursively visit nodes only if they are close enough to offer a closer point.
- **Query time:** $O(\log n + (1/\epsilon)^d)$.



BBD Tree Analysis

Query-Time Analysis

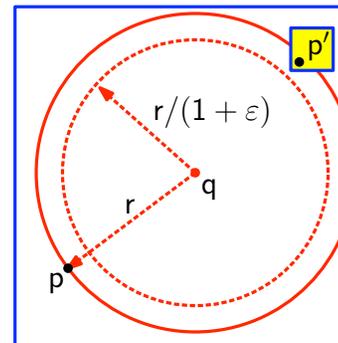
- $O(\log n)$ time to locate q .
- Let $r = \|qp\|$, where p is the closest point found by the search.
- Any cell whose distance exceeds $r/(1 + \varepsilon)$ need not be visited.
- The only cells to be split, span the annulus between these two radii.
- By a **packing argument**, the number of such cells is $O(1/\varepsilon^{d-1})$.



BBD Tree Analysis

Query-Time Analysis

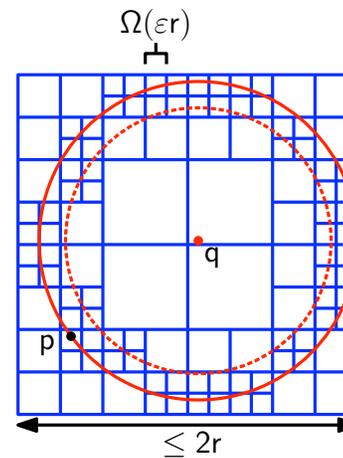
- $O(\log n)$ time to locate q .
- Let $r = \|qp\|$, where p is the closest point found by the search.
- Any cell whose distance exceeds $r/(1 + \varepsilon)$ need not be visited.
- The only cells to be split, span the annulus between these two radii.
- By a **packing argument**, the number of such cells is $O(1/\varepsilon^{d-1})$.



BBD Tree Analysis

Query-Time Analysis

- $O(\log n)$ time to locate q .
- Let $r = \|qp\|$, where p is the closest point found by the search.
- Any cell whose distance exceeds $r/(1 + \varepsilon)$ need not be visited.
- The only cells to be split, span the annulus between these two radii.
- By a **packing argument**, the number of such cells is $O(1/\varepsilon^{d-1})$.



Euclidean Spaces — ϵ -Dependencies

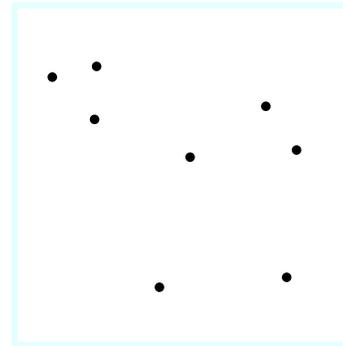
Reducing ϵ -Dependencies

- Space is $O(n)$.
⇒ **Great!** No exponential dependence on dimension.
- Query time is $O(\log n + (1/\epsilon)^{d-1})$.
⇒ The $O(1/\epsilon)^{d-1}$ term **dominates** in practice.
- Can it be reduced, perhaps at the expense of greater space?
- Can we offer **space-time tradeoffs**?

Euclidean Spaces - AVDs

Approximate Voronoi Diagram (AVD)

- Introduced by Har-Peled [Har01].
- Partition the space into **cells**.
- **Quadtree** for fast point location.
- Each cell stores a **representative**, an ϵ -NN of any point in the cell.

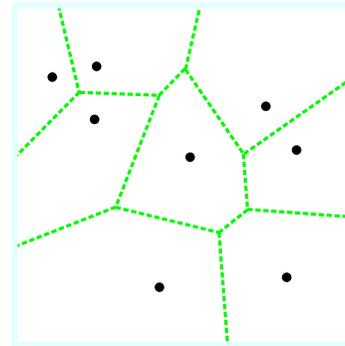


- **Query time:** $O(\log(n/\epsilon))$ (**Fast!**)
- **Space:** $\tilde{O}(n/\epsilon^{d-1})$ [Har01]

Euclidean Spaces - AVDs

Approximate Voronoi Diagram (AVD)

- Introduced by Har-Peled [Har01].
- Partition the space into **cells**.
- **Quadtree** for fast point location.
- Each cell stores a **representative**, an ϵ -NN of any point in the cell.

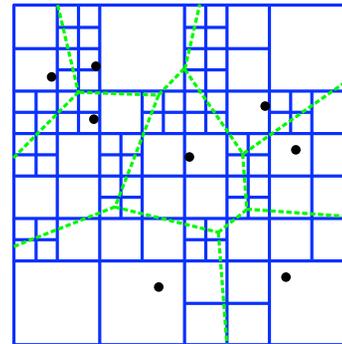


- **Query time:** $O(\log(n/\epsilon))$ (**Fast!**)
- **Space:** $\tilde{O}(n/\epsilon^{d-1})$ [Har01]

Euclidean Spaces - AVDs

Approximate Voronoi Diagram (AVD)

- Introduced by Har-Peled [Har01].
- Partition the space into **cells**.
- **Quadtree** for fast point location.
- Each cell stores a **representative**, an ϵ -NN of any point in the cell.



- **Query time:** $O(\log(n/\epsilon))$ (Fast!)
- **Space:** $\tilde{O}(n/\epsilon^{d-1})$ [Har01]

Euclidean Spaces - AVDs

Approximate Voronoi Diagram (AVD)

- Introduced by Har-Peled [Har01].
- Partition the space into **cells**.
- **Quadtree** for fast point location.
- Each cell stores a **representative**, an ϵ -NN of any point in the cell.

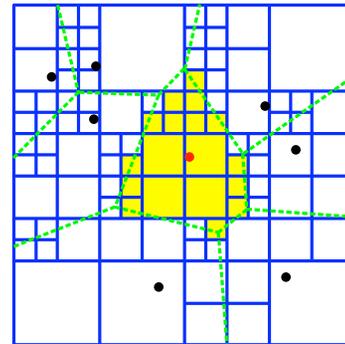
- **Query time:** $O(\log(n/\epsilon))$ (Fast!)
- **Space:** $\tilde{O}(n/\epsilon^{d-1})$ [Har01]



Euclidean Spaces - AVDs

Approximate Voronoi Diagram (AVD)

- Introduced by Har-Peled [Har01].
- Partition the space into **cells**.
- **Quadtree** for fast point location.
- Each cell stores a **representative**, an ϵ -NN of any point in the cell.



- **Query time:** $O(\log(n/\epsilon))$ (Fast!)
- **Space:** $\tilde{O}(n/\epsilon^{d-1})$ [Har01]

Existing Results

Ignoring logarithmic factors:

Space	Time	Method	Product
n	$1/\epsilon^{d-1}$	BBD-trees [AMN98]	n/ϵ^{d-1}
$n/\epsilon^{\frac{d-1}{2}}$	$1/\epsilon^{\frac{d-1}{2}}$	Polytope Approx. [Cla94, Cha98]	n/ϵ^{d-1}
n/ϵ^{d-1}	1	AVDs [Har01, AMM09]	n/ϵ^{d-1}

Space-Time Tradeoffs

Existing Results

Ignoring logarithmic factors:

Space	Time	Method	Product
n	$1/\varepsilon^{d-1}$	BBD-trees [AMN98]	n/ε^{d-1}
$n/\varepsilon^{\frac{d-1}{2}}$	$1/\varepsilon^{\frac{d-1}{2}}$	Polytope Approx. [Cla94, Cha98]	n/ε^{d-1}
n/ε^{d-1}	1	AVDs [Har01, AMM09]	n/ε^{d-1}

- A pattern emerges...
- **Space:** $M_\epsilon(n)$
 - **Query Time:** $T_\epsilon(n)$
 - **Tradeoff:** $M_\epsilon(n)T_\epsilon(n) = O(n/\epsilon^{d-1})$.

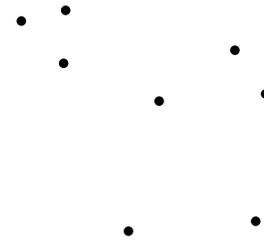
Challenge: Can we find a single **unified approach** that achieves the tradeoff, throughout the **entire spectrum**?

Multiple-Representative AVD

What if we allow each cell to have **multiple representatives**?

(t, ϵ) -AVD

- Subdivision by BBD tree cells.
- Associate each cell with at most t **representatives**, such that for any point in the cell, one of these is an ϵ -NN.



Allowing more representatives reduces the number of cells **significantly**.

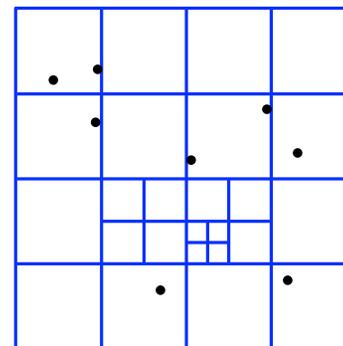
Multiple-Representative AVD

What if we allow each cell to have **multiple representatives**?

(t, ε) -AVD

- Subdivision by BBD tree cells.
- Associate each cell with at most t **representatives**, such that for any point in the cell, one of these is an ε -NN.

Allowing more representatives reduces the number of cells **significantly**.



(3,0)-AVD

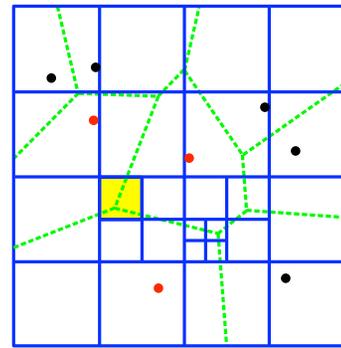
Multiple-Representative AVD

What if we allow each cell to have **multiple representatives**?

(t, ϵ) -AVD

- Subdivision by BBD tree cells.
- Associate each cell with at most t **representatives**, such that for any point in the cell, one of these is an ϵ -NN.

Allowing more representatives reduces the number of cells **significantly**.



(3,0)-AVD

Space-Time Tradeoffs

Space-Time Tradeoffs for AVDs (Arya, et al. [AMM09])

Given a tradeoff parameter γ , $2 \leq \gamma \leq 1/\varepsilon$, there is an AVD achieving $M_\varepsilon(n) = O(n\gamma^{d-1})$ and $T_\varepsilon(n) = \tilde{O}(1/(\varepsilon\gamma)^{(d-1)/2})$.

Hence, we have

$$M_\varepsilon(n)T_\varepsilon^2(n) = \frac{n}{\varepsilon^{d-1}}$$

- **Remarkably:** This is a **better** space-time tradeoff than expected! Decreasing space by $1/2$, increases query time by only $\sqrt{2}$, not 2.
- **Lower bounds:** We also have lower bounds showing that the tradeoff is tight in the extremes $\gamma = 2$ and $\gamma = 1/\varepsilon$.

Introduction ○○○○	Euclidean Spaces ○○○○○○○○○○○○	Metric Spaces ●○○○○○○○○	Almost Done ○○○
----------------------	----------------------------------	----------------------------	--------------------

Moving to Metric Spaces

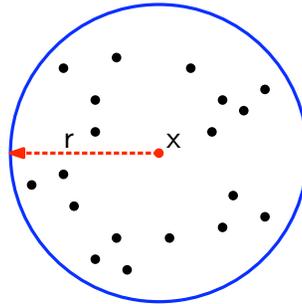
Okay, Euclidean space seems to be pretty well understood.

Next Big Question

Can these results/techniques be generalized to general **metric spaces**?

First, we need to define the concept of **dimension** for metric spaces.

Doubling Spaces



Given a metric space (S, d) , define a **ball** of radius $r > 0$ centered at x :
 $B(x, r) = \{y \in S : d(x, y) \leq r\}$.

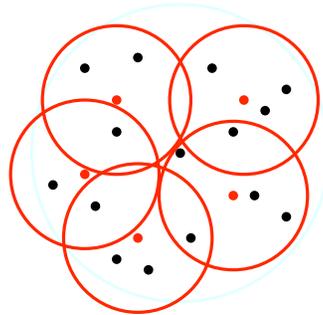
Doubling dimension \dim [Assouad 1983]

... is the minimum value ρ such that every ball in the space can be covered by 2^ρ balls of half the radius.

Doubling space

... is a metric space of constant doubling dimension.

Doubling Spaces



Given a metric space (S, d) , define a **ball** of radius $r > 0$ centered at x :
 $B(x, r) = \{y \in S : d(x, y) \leq r\}$.

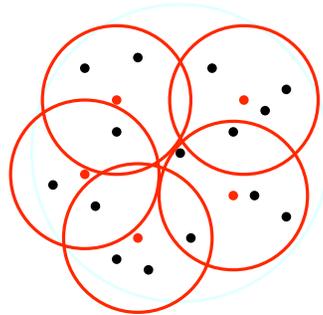
Doubling dimension \dim [Assouad 1983]

... is the minimum value ρ such that every ball in the space can be covered by 2^ρ balls of half the radius.

Doubling space

... is a metric space of constant doubling dimension.

Doubling Spaces



Given a metric space (S, d) , define a **ball** of radius $r > 0$ centered at x :
 $B(x, r) = \{y \in S : d(x, y) \leq r\}$.

Doubling dimension \dim [Assouad 1983]

... is the minimum value ρ such that every ball in the space can be covered by 2^ρ balls of half the radius.

Doubling space

... is a metric space of constant doubling dimension.

Results comparable to those for BBD trees exist in doubling spaces.

ε-NN Searching in Doubling Spaces [HM06, CG06]

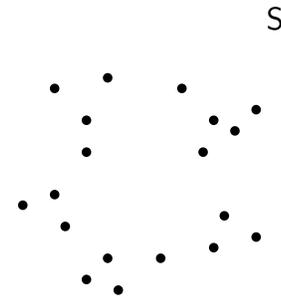
Given an n -element point set in a doubling space, it is possible to answer ε-NN queries in time $O(\log n) + (1/\varepsilon)^{O(1)}$ from a data structure of $O(n)$ space.

r -Net

r -Net

... is a subset $X \subseteq S$ such that

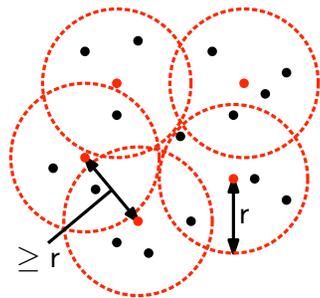
- (i) every point of S is within distance r of some point of X
- (ii) the pairwise distance between any two points of X is $\geq r$



r-Net

r-Net
... is a subset $X \subseteq S$ such that

- (i) every point of S is within distance r of some point of X
- (ii) the pairwise distance between any two points of X is $\geq r$

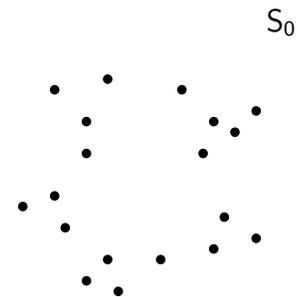


Net Tree

Net Tree

- The leaves are $S_0 = S$.
- Let r be the minimum distance between any two points of S .
- Let S_1 be an r -net of S_0 .
- Let S_2 be a $(2r)$ -net of S_1 .
- Let S_j be a $(2^j r)$ -net of S_{j-1} .
- ...
- Until only one remains — the root.

Resulting structure is the **net-tree**.

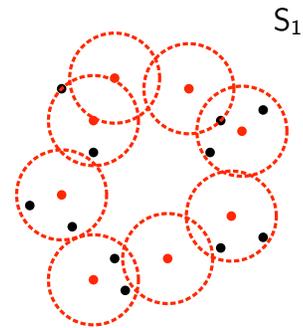


Net Tree

Net Tree

- The leaves are $S_0 = S$.
- Let r be the minimum distance between any two points of S .
- Let S_1 be an r -net of S_0 .
- Let S_2 be a $(2r)$ -net of S_1 .
- Let S_j be a $(2^j r)$ -net of S_{j-1} .
- ...
- Until only one remains — the root.

Resulting structure is the **net-tree**.

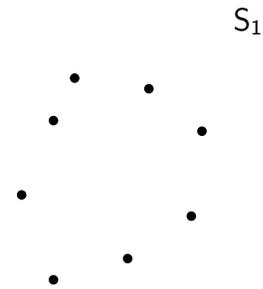


Net Tree

Net Tree

- The leaves are $S_0 = S$.
- Let r be the minimum distance between any two points of S .
- Let S_1 be an r -net of S_0 .
- Let S_2 be a $(2r)$ -net of S_1 .
- Let S_j be a $(2^j r)$ -net of S_{j-1} .
- ...
- Until only one remains — the root.

Resulting structure is the **net-tree**.

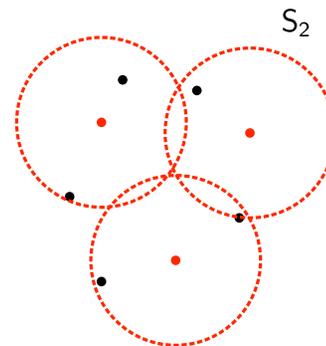


Net Tree

Net Tree

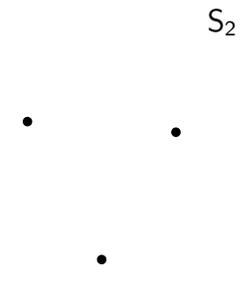
- The leaves are $S_0 = S$.
- Let r be the minimum distance between any two points of S .
- Let S_1 be an r -net of S_0 .
- Let S_2 be a $(2r)$ -net of S_1 .
- Let S_j be a $(2^j r)$ -net of S_{j-1} .
- ...
- Until only one remains — the root.

Resulting structure is the **net-tree**.



Net Tree

- Net Tree
- The leaves are $S_0 = S$.
 - Let r be the minimum distance between any two points of S .
 - Let S_1 be an r -net of S_0 .
 - Let S_2 be a $(2r)$ -net of S_1 .
 - Let S_j be a $(2^j r)$ -net of S_{j-1} .
 - ...
 - Until only one remains — the root.
- Resulting structure is the **net-tree**.

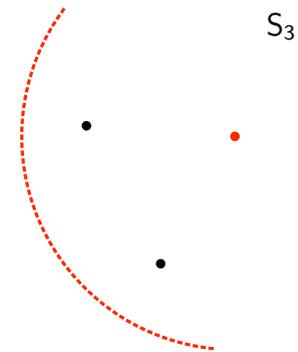


Net Tree

Net Tree

- The leaves are $S_0 = S$.
- Let r be the minimum distance between any two points of S .
- Let S_1 be an r -net of S_0 .
- Let S_2 be a $(2r)$ -net of S_1 .
- Let S_j be a $(2^j r)$ -net of S_{j-1} .
- ...
- Until only one remains — the root.

Resulting structure is the **net-tree**.



Net Tree

Net Tree

- The leaves are $S_0 = S$.
- Let r be the minimum distance between any two points of S .
- Let S_1 be an r -net of S_0 .
- Let S_2 be a $(2r)$ -net of S_1 .
- Let S_j be a $(2^j r)$ -net of S_{j-1} .
- ...
- Until only one remains — the root.

Resulting structure is the **net-tree**.

S_3

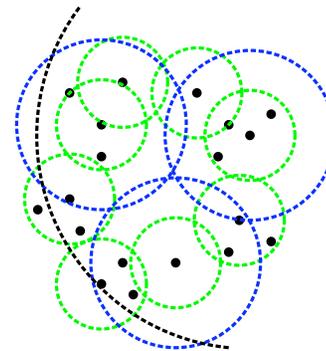


Net Tree

Net Tree

- The leaves are $S_0 = S$.
- Let r be the minimum distance between any two points of S .
- Let S_1 be an r -net of S_0 .
- Let S_2 be a $(2r)$ -net of S_1 .
- Let S_j be a $(2^j r)$ -net of S_{j-1} .
- ...
- Until only one remains — the root.

Resulting structure is the **net-tree**.



No Space-Time Tradeoffs?

Lower bounds [HM06]

There exists an n -element point set in a doubling space that requires query time $\Omega(\log n) + (1/\varepsilon)^{\Omega(1)}$ (irrespective of the space used) in the decision tree model.

This **kills** any hope of space-time tradeoffs.

Question

Is there any hope of achieving results in general doubling spaces comparable to the best results for Euclidean spaces?

Doubling Oracle and Weakly Explicit Model

Yes! But with a slightly stronger model.

Doubling Oracle

Consider any doubling space of dimension d . Given a ball b of radius r a set of at most $2^{O(d)}$ balls of radius $r/2$ covering b is returned in constant time.

Weakly explicit model

- We are given a doubling space with a **doubling oracle**.
- The points used in the data structure can be drawn from the **ambient metric space**.

Main Result

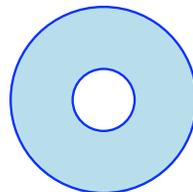
Theorem

Let S be a set of n points in the doubling space. Given a parameter $2 \leq \gamma \leq 1/\varepsilon$, we can construct a data structure of $n\gamma^{O(1)} \log(1/\varepsilon)$ space, which can answer ε -approximate nearest neighbor queries in time $O(\log(n\gamma)) + 1/(\varepsilon\gamma)^{O(1)}$.

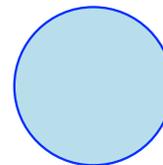
- $\gamma = 1/\varepsilon$: $O(\log(n/\varepsilon))$ query time, $n/\varepsilon^{O(1)}$ space
⇒ matches the best query times known in Euclidean spaces.
- $\gamma = 2$: $O(\log n) + (1/\varepsilon)^{O(1)}$ query time, $O(n \log(1/\varepsilon))$ space
⇒ nearly matches the best results for both Euclidean spaces and doubling spaces.

New Data Structure: Region-DAG

- Unlike quadtree-based decomposition, the region-DAG is not a hierarchical partition but a hierarchical **covering** scheme.
- Each node is associated with a region of space called a **cell**, which is the difference of two concentric balls, an **outer ball** and an (optional) **inner ball**.



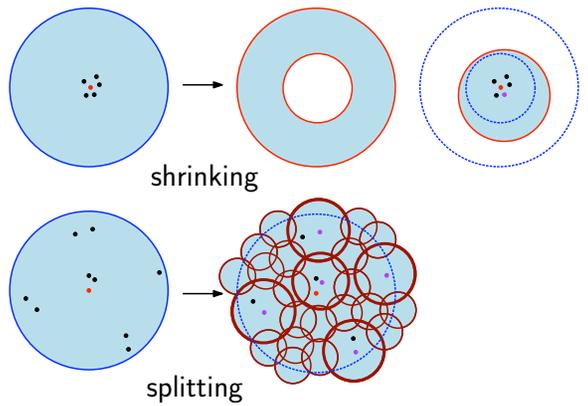
Doughnut cell



Simple cell

Region-DAG: Internal Nodes

The cell associated with an internal node is always simple, and is covered by the cells associated with its children.



- We have surveyed many of the results on **approximate nearest neighbor searching**, both in **Euclidean spaces** of constant dimension and space of constant **doubling dimension**.
- We have shown how to **generalize AVDs** to doubling spaces, which enable us to achieve much **faster query times** than previously possible, and also the **space-time tradeoffs**.
- Net trees hold promise for many other geometric retrieval problems — retrieval involving **points in motion**.
- What **other geometric search problems** can be generalized from Euclidean space to doubling spaces? — Range searching?

- Collaborators:
 - Sunil Arya (HKUST, Hong Kong),
 - Guilherme da Fonseca (UNIRIO, Rio De Janeiro),
 - Charis Malamatos (University of Peloponnese, Greece),
 - Antoine Vigneron (INRA, Paris).
- Funding Organizations:
 - National Science Foundation,
 - Office of Naval Research.



Thank you!
Merci!
Gracias!
Obrigado!
Mamnoon!
Grazie!
Danke!

Bibliography

-  [S. Arya, D. M. Mount, N. Netanyahu, R. Silverman, and A. Y. Wu.](#)
An optimal algorithm for approximate nearest neighbor searching in fixed dimensions.
J. ACM 45, 891–923, 1998.
-  [S. Arya, T. Malamatos, and D. M. Mount.](#)
Space-time tradeoffs for approximate nearest neighbor searching.
J. ACM 57, 2009 (to appear).
-  [T. M. Chan.](#)
Approximate nearest neighbor queries revisited.
Discrete Comput. Geom. 20, 359–373, 1998.
-  [K. L. Clarkson,](#)
An algorithm for approximate closest-point queries.
In *Proc. 10th Annu. ACM Sympos. Comput. Geom.* 160–164, 1994.
-  [R. Cole and L. Gottlieb.](#)
Searching dynamic point sets in spaces with bounded doubling dimension.
In *Proc. 38th Annu. ACM Sympos. Theory Comput.*, 574–583, 2006.
-  [S. Har-Peled.](#)
A replacement for Voronoi diagrams of near linear size.
In *Proc. 42nd Annu. IEEE Sympos. Found. Comput. Sci.*, 94–103, 2001.
-  [S. Har-Peled and M. Mendel.](#)
Fast construction of nets in low dimensional metrics, and their applications.
SIAM J. Comput., 35(5):1148–1184, 2006.