Approximate Voronoi Diagrams: Techniques, tools, and applications to kth ANN search

Nirman Kumar

University of California, Santa-Barbara

January 13th, 2016

Similarity Search

?

Need similarity search to make sense of the world!

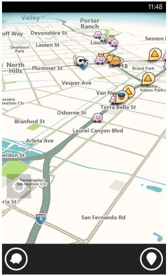
When an appropriate metric is defined

Similarity search reduces to NN search

Nearest neighbor search

Set of points P: find quickly for a query q, the closest point to q in P

Nearest neighbor search



Also important in other domains

Approximate nearest neighbor search (ANN)

Find any point x with

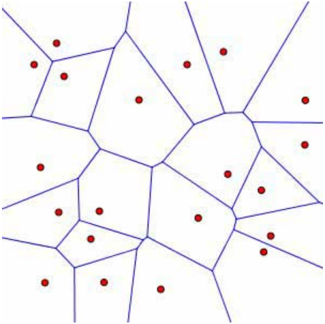
$$d(q, x) \leq (1 + \varepsilon) d_1(q, P)$$

Most data structures for NN (or ANN) search partition space

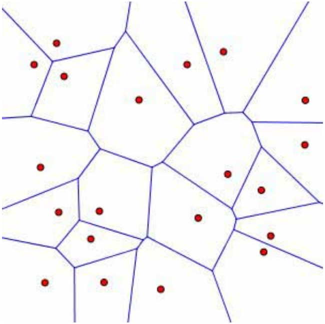
In low dimensions this is an explicit paritioning

In high dimensions the partitioning is implicit (via hash functions)

Voronoi diagrams

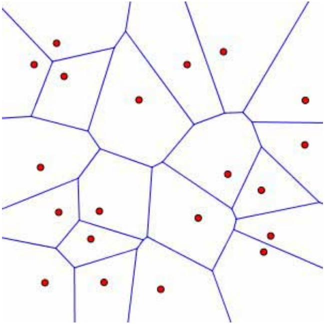


Voronoi diagrams



Very efficient in dimensions $d \leq 2$

Voronoi diagrams



Performance degrades sharply - bad even for $d = 3$

This talk

- **Construction of Approximate Voronoi Diagrams**
- **Tools used - Quadtrees, WSPD**
- **Construction of AVD for kth ANN**
- **Some open problems**

A space partition as before

With each region is associated 1 rep (a point of P)

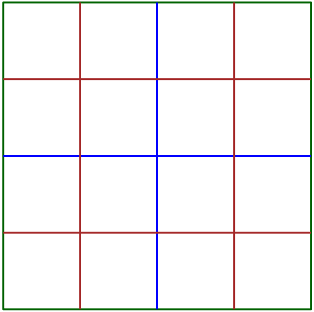
Approximate Voronoi Diagrams (AVD)

This rep is a valid ANN for any q in region

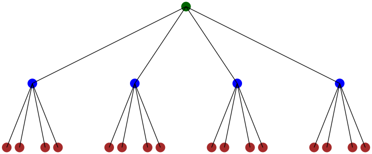
Main ideas behind ANN search and AVDs

- **If the query point is "far" *any* point is a good ANN**
- **A region can be approximated well by cubes**
- **Point location can be done in a set of cubes *efficiently***

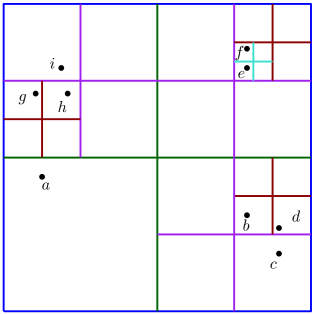
A quadtree - intuitively

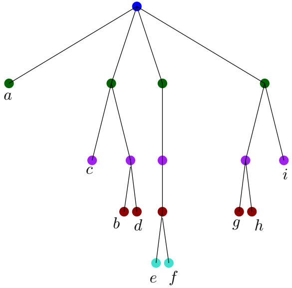


$[0,1] \times [0,1]$

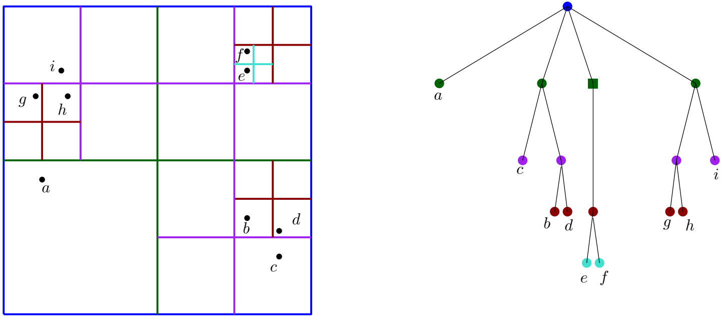


A quadtree on points





The compressed version



Point Location $\equiv$ find leaf node containing a point

Height h: $O(\log h)$ time - $O(\log \log n)$ for balanced tree!

But height not bounded as function of n

Tool 1: Quadtrees

Use compressed quadtree - height bounded by $O(n)$

Tool 2: Well separated pairs decomposition

How many distances among points - $\Omega(n^2)$

What if distances within $(1 \pm \varepsilon)$ are considered the same?

Tool 2: Well separated pairs decomposition

About $O(n/\varepsilon^d)$ different distinct distances upto $(1 \pm \varepsilon)$

Tool 2: Well separated pairs decomposition

- **How can we represent them?**

- **Given a pair of points, which bucket does it belong to?**

Tool 2: Well separated pairs decomposition

The WSPD data structure captures this

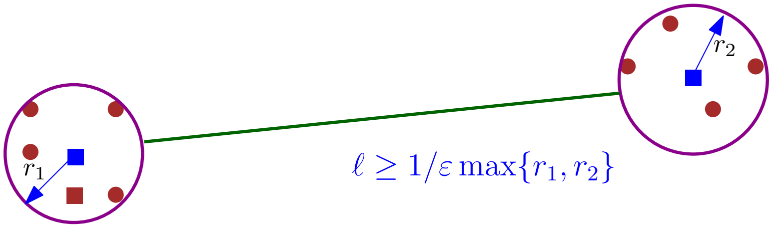
Tool 2: Well separated pairs decomposition

More formally

- **A collection of pairs** $A_i, B_i \subset P$
- $A_i \cap B_i = \emptyset$
- **Every pair of points is separated by some** A_i, B_i
- **Each pair** A_i, B_i **is well separated**

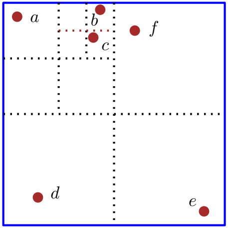
Tool 2: Well separated pairs decomposition

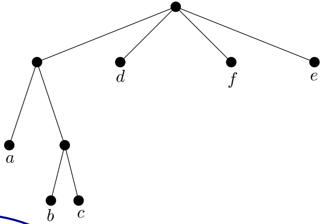
A well separated pair is a dumbbell

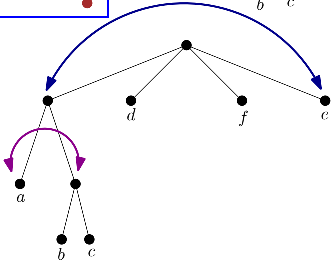


Tool 2: Well separated pairs decomposition

WSPD example







$A_1 = \{a, b, c\}, B_1 = \{e\}$

$A_1 = \{a\}, B_1 = \{b, c\}$

$\vdots$

Main result about WSPDs

There is a ε^{-1}-WSPD of size $O(n\varepsilon^{-d})$ - It can be constructed in $O(n \log n + n\varepsilon^{-d})$ time

The main result

- $O(n/\varepsilon^d)$ **cells**
- **Query time -** $O(\log(n/\varepsilon))$

Construct a 8-WSPD for the point set

Let (A_i, B_i) for $i = 1, \ldots, m$ be the pairs

For each pair do some processing - output some cells

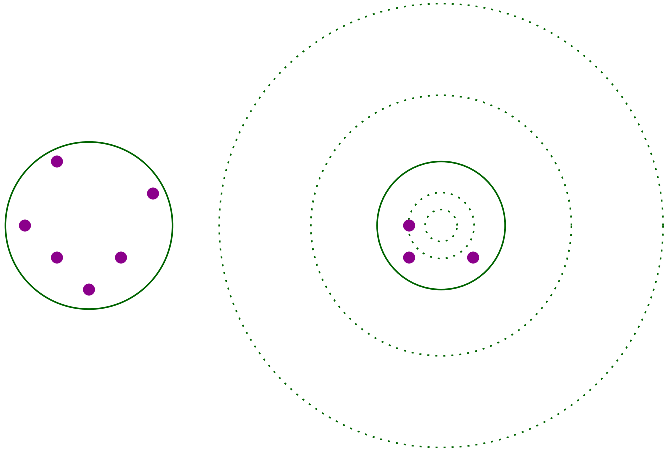
Preprocess them for point location

So what is the processing per pair?

The AVD algorithm

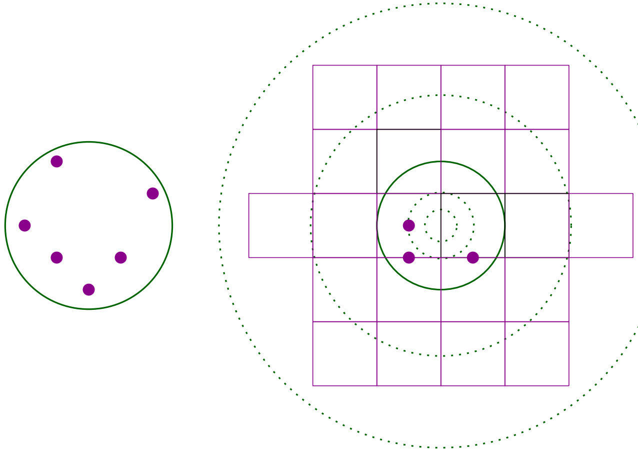
Consider a WSPD dumbbell

The AVD algorithm



Concentric balls increasing radii - $r/4$ to $\approx r/\varepsilon$

The AVD algorithm



Tile each ball (rad x) by cubes of size $\approx \varepsilon x$

Store the ε/c ANN for some point in each cell

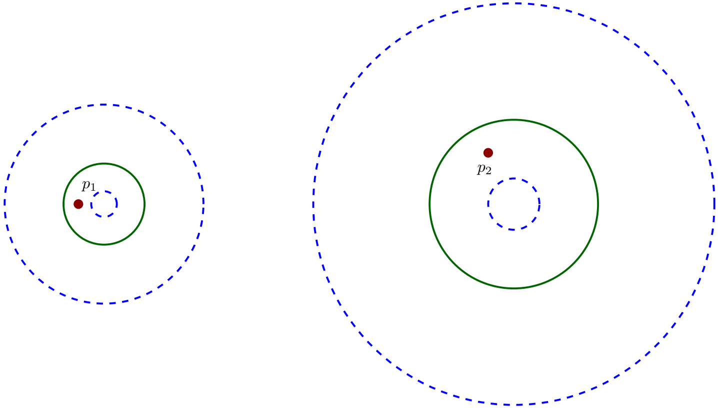
So why does it work?

Every pair* of competing points is *resolved

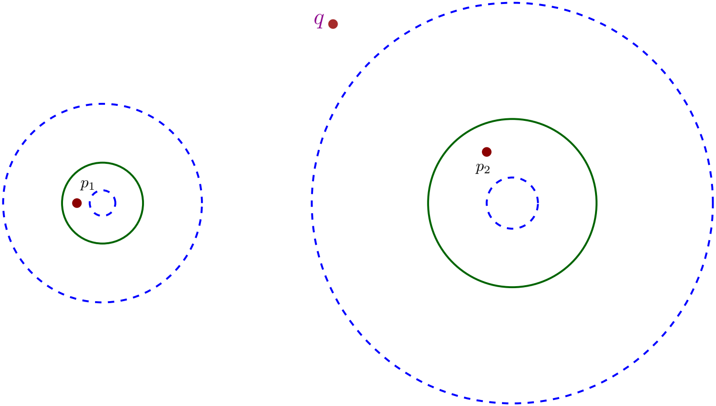
So why does it work?

p_1, p_2 resolved by the WSPD pair separating them

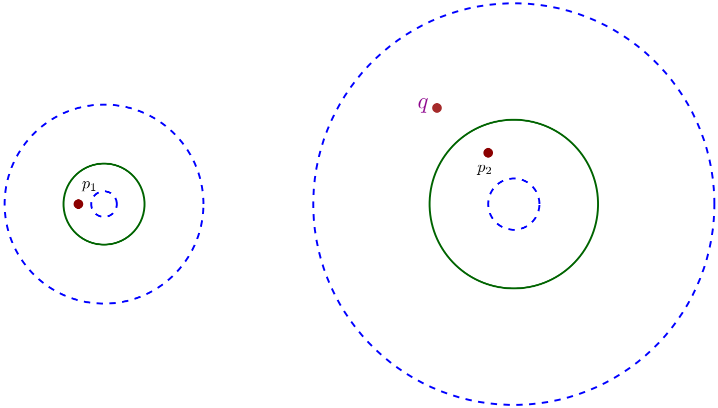
So why does it work?



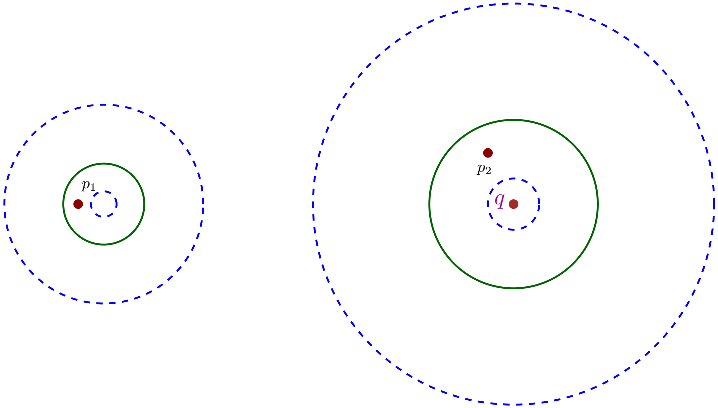
So why does it work?



So why does it work?



So why does it work?



Bounding the AVD complexity

The shown method gives $O(n/\varepsilon^d \log 1/\varepsilon)$ cubes

Bounding the AVD complexity

This can be improved to $O(n/\varepsilon^d)$

kth ANN search

Given q output a point $u \in P$ such that:

$$(1 - \varepsilon) d_k(q, P) \leq d(q, u) \leq (1 + \varepsilon) d_k(q, P)$$

Applications of kth ANN search

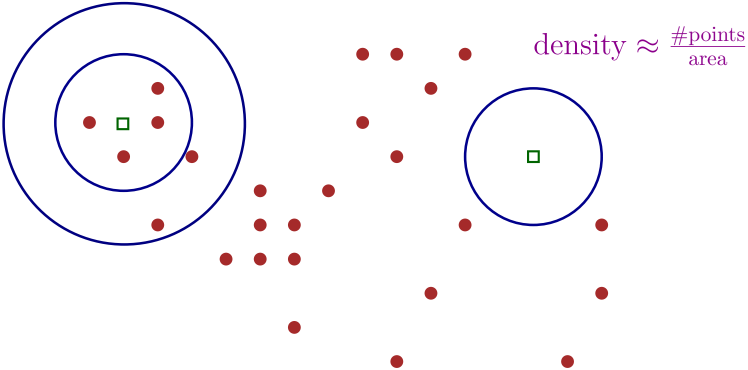
- **Density estimation**

- **Functions of the form :** $F(q) = \sum_{i=1}^{k} f(d_i(q, P))$

- k**th ANN on balls**

Applications of kth ANN search

Density estimation

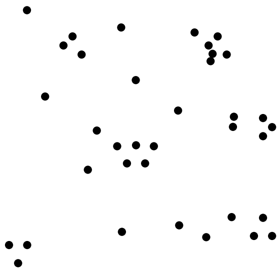


The result

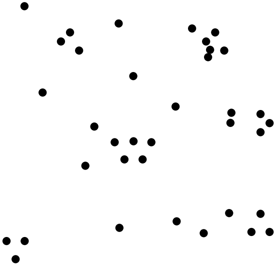
AVD for kth ANN

- $O((n/k)\varepsilon^{-d}\log 1/\varepsilon)$ **cells**
- **Query time -** $O(\log(n/(k\varepsilon)))$

Quorum clustering

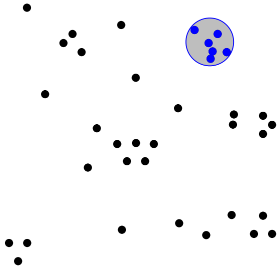


Quorum clustering



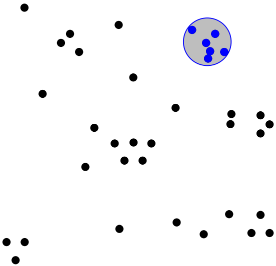
Find smallest ball containing k points

Quorum clustering



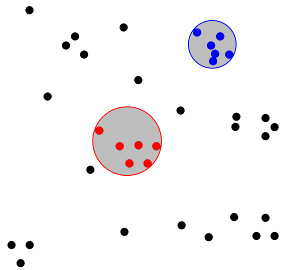
Find smallest ball containing k points

Quorum clustering



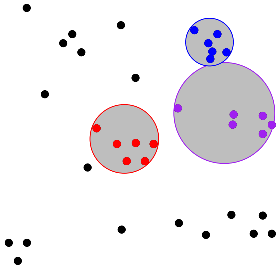
Remove points and repeat

Quorum clustering



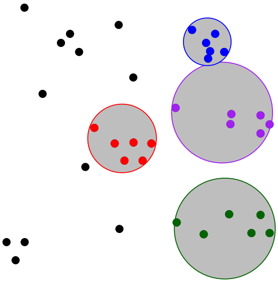
Remove points and repeat

Quorum clustering



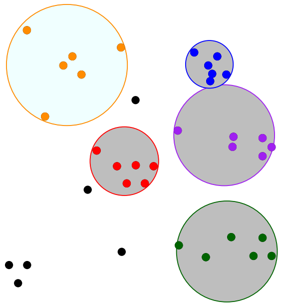
Remove points and repeat

Quorum clustering



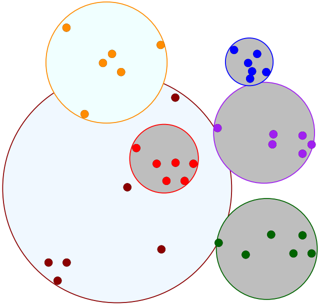
Remove points and repeat

Quorum clustering



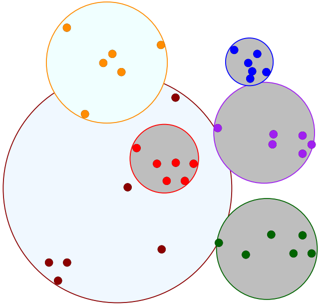
Remove points and repeat

Quorum clustering



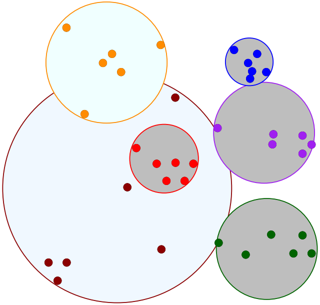
Remove points and repeat

Quorum clustering



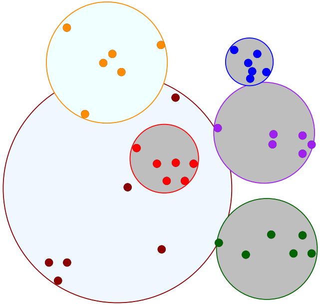
A way to summarize points

Quorum clustering



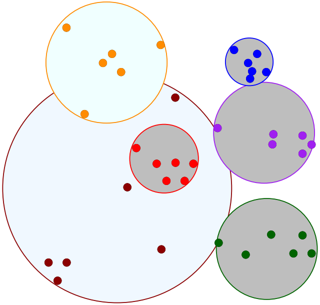
Has properties favorable for kth ANN problem

Quorum clustering



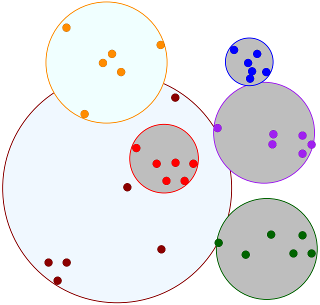
Quorum clustering too expensive to compute

Quorum clustering



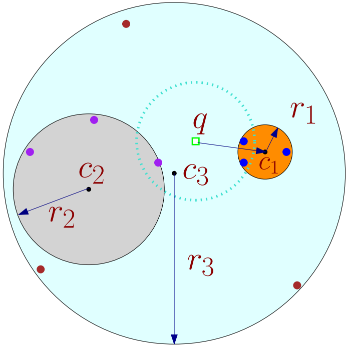
Can compute approximate quorum clustering

Quorum clustering



- **Computed in:** $O(n \log^d n)$ **time in** $\mathbb{R}^d$ **[Carmi, Dolev, Har-Peled, Katz and Segal, 2005]**
- **Computed in:** $O(n \log n)$ **time in** $\mathbb{R}^d$ **[Har-Peled and K., 2012]**

Why is quorum clustering useful

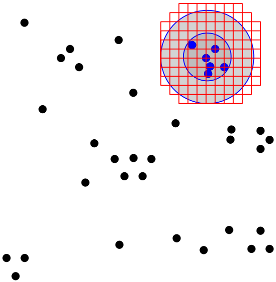


- $x = d_k(q, P)$
- $r_1 \leq x$
- $x + r_1 \geq d(q, c_1) \implies d(q, c_1) \leq 2x$
- $x \leq d(q, c_1) + r_1 \leq 3x$

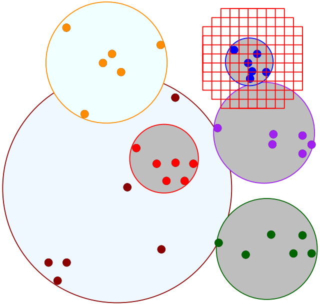
Refining the approximation

Just as in AVDs generate a list of cells

Refining the approximation



Refining the approximation



Refining the approximation

For closest ball use ANN data structure in $\mathbb{R}^{d+1}$

Refining the approximation

$$b = b(c, r) \to (c, r) \in \mathbb{R}^{d+1}$$

Some cells generated by AVD for ball centers

Refining the approximation

Store some info with each cell

A kth ANN, and approximate *closest* ball

Open problems

- **In high dimensions, is there a data structure for kth NN whose space requirement is $f(n/k)$?**

- **There is an AVD for weighted ANN similar to AVD as shown - is there an extension to weighted kth ANN?**