

Fast Construction of Nets in Low-Dimensional Metrics and Their Applications

Sariel Har-Peled, Manor Mendel

Presented by
Anthony Man-Cho So
24 May 2006

Motivation and Goal

- Carry over algorithmic results to doubling metrics.
- One way is to construct a hierarchical decomposition of the metric space.
 - Recall the Euclidean case – e.g. compressed quadtrees.

Net-Tree Basics

- In abstract metric spaces, we need nets.
 - Usually a sequence of increasingly refined subsets of P .
 - At a given resolution, there is an element in the sequence that represents the structure of P in that resolution.
- We will consider the construction of a net-tree.

Net-Tree Basics

- Formally, let P be an n -point metric space. A net-tree for P is a tree T s.t. :
 - the set of leaves is P
 - each node u of T has a representative rep_u of P and a level $l(u)$
 - the levels satisfy $l(u) < l(p(u))$ and $l(u) = -\infty$ if u is a leaf
 - each internal node of T has at least 2 children
 - for every non-leaf $u \in T$, there exists a child v of u s.t. $rep_v = rep_u$

Net-Tree Basics

- Let P_u be the set of leaves in the subtree rooted at $u \in T$. Let b be some constant. We further require that:

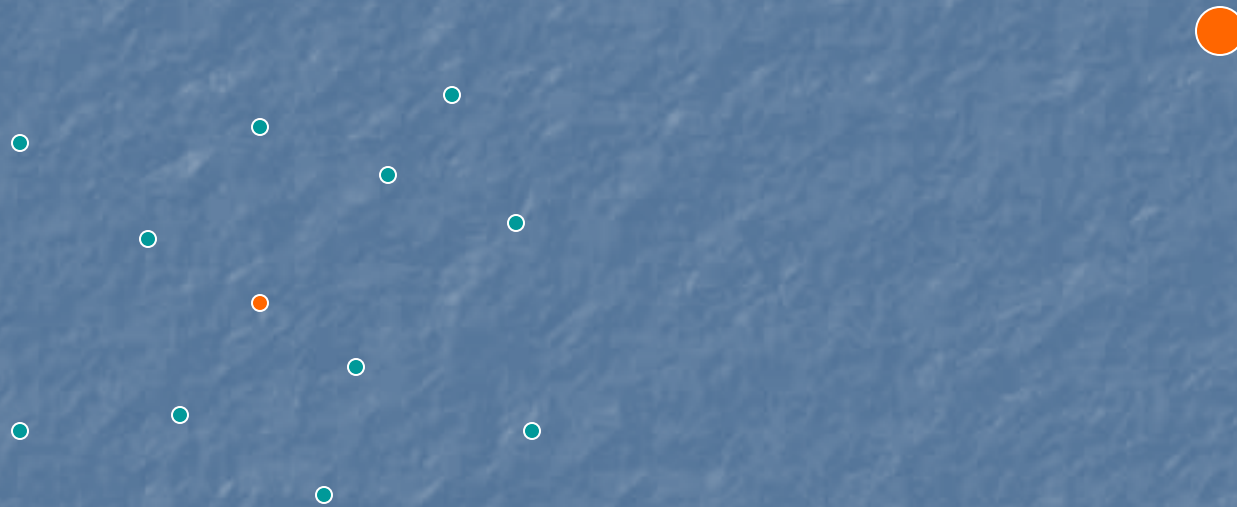
- Covering: for all $u \in T$,

$$B\left(\text{rep}_u, \frac{2b}{b-1} b^{l(u)}\right) \supseteq P_u$$

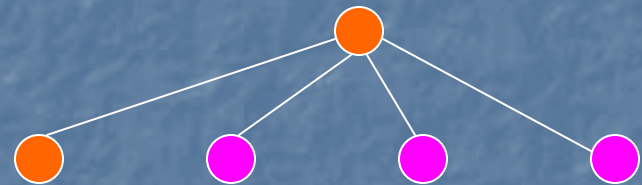
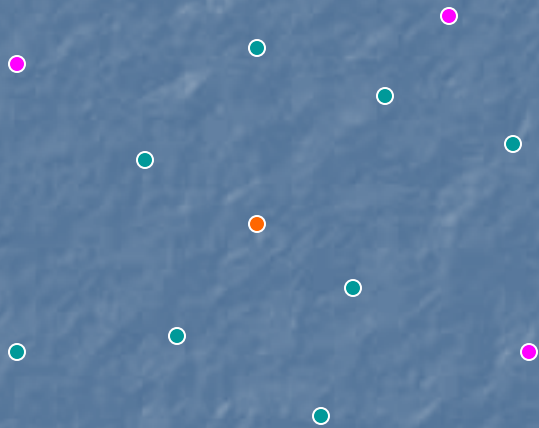
- Packing: for all non-root $u \in T$,

$$B\left(\text{rep}_u, \frac{b-5}{2(b-1)} b^{l(p(u))-1}\right) \cap P \subset P_u$$

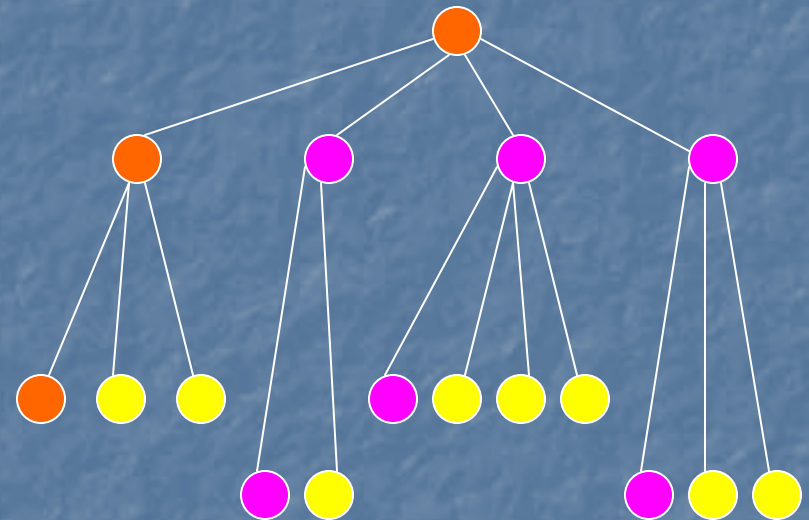
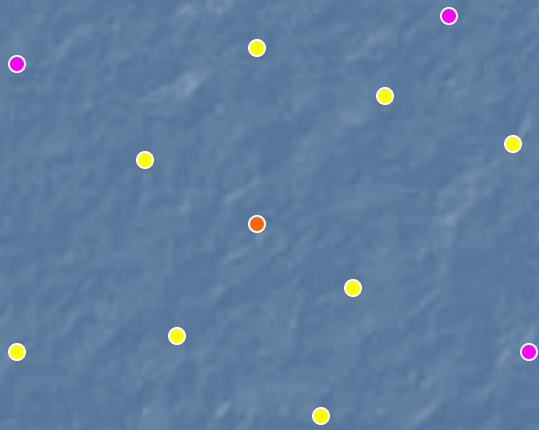
An Illustration



An Illustration



An Illustration



From Net-Tree to Net

- Let $N(l) = \{rep_u : l(u) < l \leq l(p(u))\}$.
- Claim: $N(l)$ is a net in the following sense. For every $p, q \in N(l)$, $d(p, q) \geq b^{l-1}/4$. Also, we have $P \subset \bigcup_{p \in N(l)} B(p, 4b^l)$
- Proof:
 - Consider $B_p = B(p, r_p)$, $B_q = B(q, r_q)$, where $p, q \in N(l)$; u, v are the corresponding nodes in the net-tree, respectively, and r_p, r_q are the packing radii:

$$r_p = \frac{b-5}{2(b-1)} b^{l(p(u))-1}; \quad r_q = \frac{b-5}{2(b-1)} b^{l(p(v))-1}$$

From Net-Tree to Net

■ Proof (con't):

- Now, $B_p \cap P \subseteq P_u$ and $B_q \cap P \subseteq P_v$.
- Since u, v are on different branches, and P_u and P_v are disjoint.
- Then, $d(p, q) \geq \max\{r_p, r_q\} \geq b^{l-1}/4$.
- The covering property can be proven in a similar fashion.

Doubling Dimension

- The doubling dimension of P (denoted $\dim(P)$) is the least m s.t. for all $P' \subseteq \mathbb{R}^d$, there exist Z_1, \dots, Z_{2^m} s.t. $\text{diam}(Z_i) \leq \text{diam}(P')/2$ and $P' \subset \bigcup_i Z_i$
- Let $I = 2^m$ be the doubling constant.
- The packing and covering properties imply that each $u \in T$ has $I^{O(I)}$ children.

Application: WSPD

- Recall the definition of an WSPD:
 - A collection of pairs $\{(A_1, B_1), \dots, (A_m, B_m)\}$ s.t.
 - every pair of points is represented in exactly one pair;
 - for every $i=1, \dots, m$, A_i and B_i can be respectively enclosed by two balls of diameter D , s.t. the distance between the balls is at least sD , for some global $s > 0$.

Application: WSPD

- Theorem: For $0 < \epsilon \leq 1$, an $(1/\epsilon)$ -WSPD of size $n e^{O(\dim)}$ can be constructed in expected time $2^{O(\dim)} n \log n + n e^{O(\dim)}$.
- Proof: Invoke *genWSPD* that takes a pair (u, v) of nodes of the net-tree as input.
 - Assume $l(u) < l(v)$. Else, swap them.
 - If $8 \frac{2b}{b-1} b^{l(u)} \leq \epsilon \cdot d(\text{rep}_u, \text{rep}_v)$, return $\{(u, v)\}$.
 - Else, let u_1, \dots, u_r be the children of u , and return
$$\bigcup_{i=1}^r \text{genWSPD}(u_i, v)$$

Application: WSPD

- Proof (con't): We need to check that:
 - (i) every pair of points is covered by a pair of subsets (P_u, P_v) of the algorithm;
 - (ii) the number of pairs output by the algorithm is as advertised.
- For (ii), let (u, v) be an output pair, and assume $\text{genWSPD}(u, v)$ was issued by $\text{genWSPD}(u, p(v))$. As in the Euclidean case, we charge this to $p(v)$.

Application: WSPD

■ Proof (con't):

- Fix $v' \in T$. It is charged by pairs of the form (u, v) with $p(v) = v'$.
- Since we split $p(v)$ in $\text{genWSPD}(u, p(v))$, we have $l(v') \geq l(u)$.
- Since we split $p(u)$ in $\text{genWSPD}(p(u), p(v))$, we also have $l(p(u)) \geq l(v')$.
- Since (u, v') is not generated, we have

$$8 \frac{2b}{b-1} b^{l(v')} > \varepsilon \cdot d(\text{rep}_u, \text{rep}_{v'})$$

Application: WSPD

■ Proof (con't):

■ Consider

$$U = \left\{ w : l(p(w)) \geq l(v') \geq l(w), d(rep_u, rep_{v'}) < 8 \frac{2b}{\varepsilon(b-1)} b^{l(v')} \right\}$$

■ It contains u , and U is a subset of $N(l(v'))$.

Now, for distinct $u_1, u_2 \in U$, we have $d(P_{u_1}, P_{u_2}) \geq \frac{b^{l(v')-1}}{4}$

■ Thus, by const. doubling, we have $|U| \leq \varepsilon e^{O(dim)}$.

■ Thus, v' is only charged by pairs in $U \times C(v')$, where $C(v')$ is the children of v' .

■ But $|C(v')| \leq 2^{O(dim)}$, and the tree has only $O(n)$ nodes.

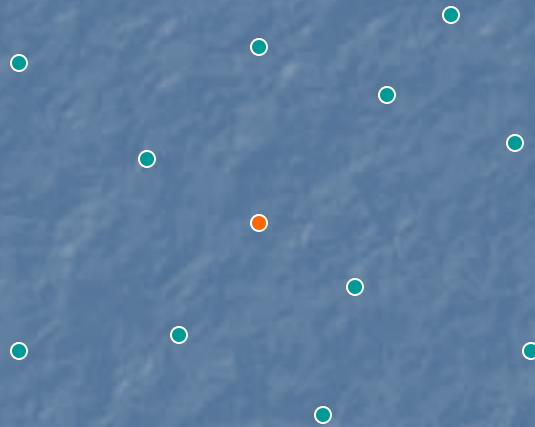
Construction of Net-Trees

- Tasks:
 - Construct a pseudo-net via a greedy clustering algorithm.
 - Turn the pseudo-net into a net-tree.
- Goal: Do all of the above **fast**. In particular, we want to remove the dependence on spread.

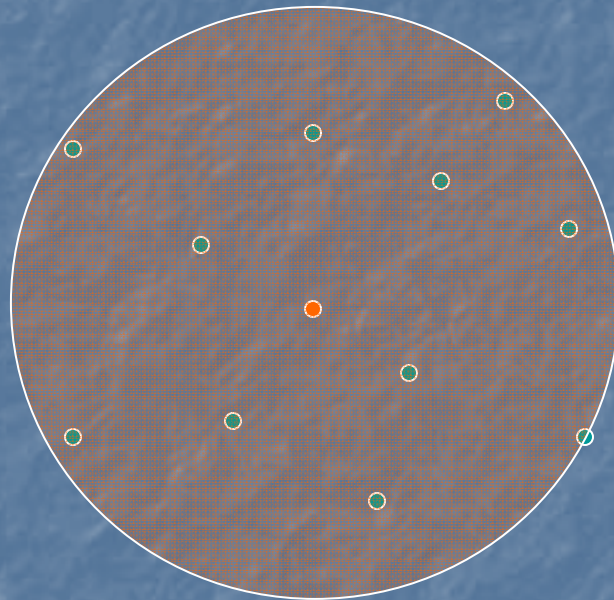
Greedy Clustering

- Gonzalez's algorithm
 - Pick an arbitrary point p_1 in P to be the first center, set $r_1 = \max_p d(p, p_1)$.
 - In the k -th iteration, for $q \in P$, let $\alpha_q^k = \min_{1 \leq i \leq k-1} d(q, p_i)$
 - The k -th center is then $p_k = \arg \max_{q \in P} \alpha_q^k$
- This yields a permutation p_1, \dots, p_n of P and numbers $r_1 \geq \dots \geq r_n = 0$ s.t. $P \subset \bigcup_{i=1}^k B(p_i, r_k)$
- In particular, it has some version of the covering property.

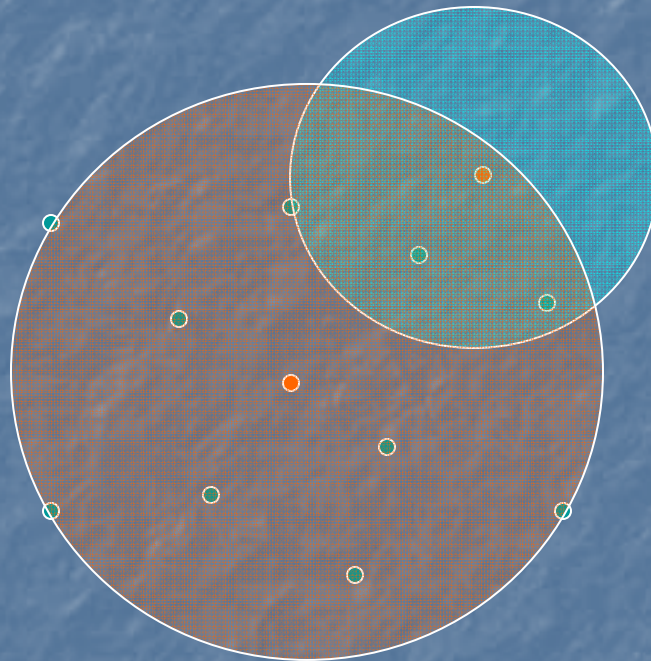
Greedy Clustering: Illustration



Greedy Clustering: Illustration



Greedy Clustering: Illustration



Greedy Clustering

- Question: How to get a fast implementation of Gonzalez's algorithm?
- Define a *phase* of the algorithm as follows. A phase starting at the i -th iteration ends at the j -th iteration if $r_j \leq r_i/2$.

Greedy Clustering

- We maintain 4 data structures:
 - for each $p \in P$ the center that serves it;
 - a max-heap that maintains, for each center p_i , the point $p_i' \in P$ that is furthest from it, as well as its a value;
 - for each center p_i , maintain a friends list that contains all the centers within $6r_k$ from p_i at the end of the k -th iteration;
 - for each center p_i , maintain its serving center 2 phases ago, together with its friends list at that time.

Greedy Clustering: Observations

- Some quick observations:
 - The size of the friends list is $O(1)$.
 - By definition, $r_{k-1} \geq \alpha_q^k = \min_{1 \leq i \leq k-1} d(q, p_i)$. Thus, if $d(q, p_k) > r_{k-1}$, then $\alpha_q^{k+1} = \alpha_q^k$. In particular, we only need to consider those $q \in P$ that are within r_{k-1} from p_k when updating α_q^{k+1} .

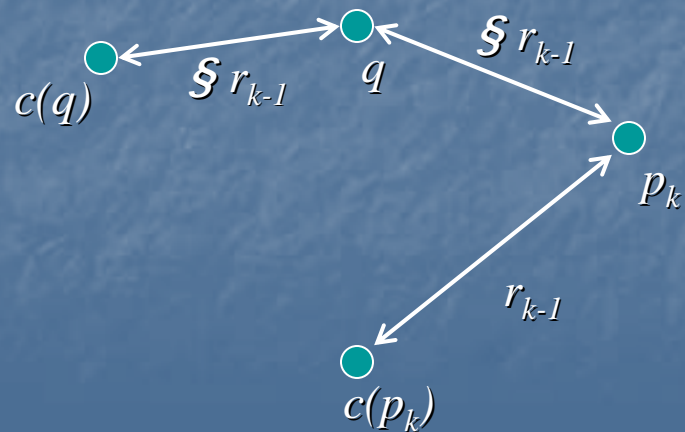
Greedy Clustering: Update Rules

- (Find the New Center) Extract from the max-heap, set p_k to be the new center. Let $c(p_k)$ be its previous serving center.
- (Update a) Scan the points currently served by $c(p_k)$ or by the centers in the friends list of $c(p_k)$ and update if necessary.
 - Why is this sufficient?

Greedy Clustering: Update Rules

- We need to show that all $q \in P$ with $d(q, p_k) \leq r_{k-1}$ are scanned by the algorithm.
- If $q \in P$ is served by $c(p_k)$, we are done.
- Otherwise, let $c_q \in \{p_1, \dots, p_{k-1}\}$ be the center serving q just before p_k is chosen.
- We then have

$$d(c(q), c(p_k)) \leq 3r_{k-1}$$



Greedy Clustering: Update Rules

- (Update Clusters) Move the relevant points in P to the new cluster and update p_i ' for each p_i in the friends list of $c(p_k)$.
- (Update Friends Lists) Scan the old friends list of p_k " (i.e. the center 2 phases ago):
 - If a scanned point p_j is within $6p_k$ of p_k , then add p_j to p_k 's friends list and vice versa.
 - For each p_j in the friends list of p_k " , scan the current friends list of p_j and update accordingly.

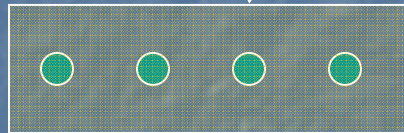
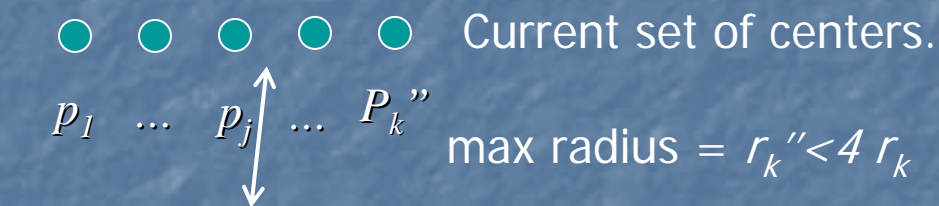
Greedy Clustering: Update Rules

- Why is this sufficient? We need to show that the set $S = \{p_i : i < k, d(p_i, p_k) \leq 6r_k\}$ is scanned. The proof is by induction.
- Let $p_i \in S$, and let r_k'' be the radius 2 phases ago. Then, $2r_k \leq r_k'' < 4r_k$
- If p_i is added before p_k'' , then we are done, since $d(p_i, p_k'') \leq d(p_i, p_k) + d(p_k, p_k'') \leq 6r_k + r_k'' \leq 6r_k''$
- O/W, we need to show that there exists a center p_j added before p_k'' (or is equal to p_k'') s.t. the friends list of p_j contains p_i .

Greedy Clustering: Update Rules

- The following guarantees the existence of

p_j .



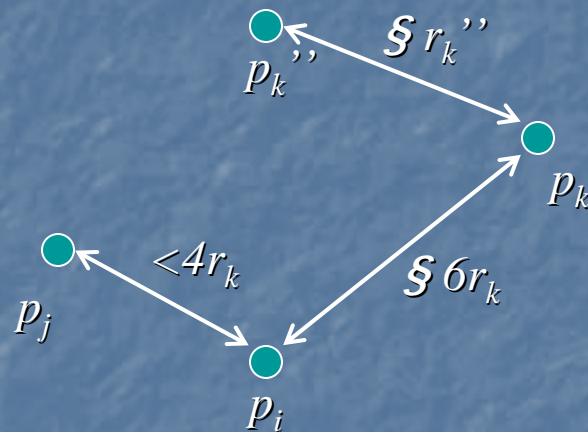
All remaining points' respective serving centers are at most r_k'' away.

- By induction, the friends list of p_j contains

p_i .

Greedy Clustering: Update Rules

- But then we have the following situation:



- Hence, we have $d(p_i, p_k'') \approx 6r_k''$ as desired.
- Of course we still need to shrink the friends lists, but this can be done in a lazy fashion.

Greedy Clustering: Runtime

- Consider $q \in P$. It is scanned in the k -th iteration if it is within $7r_k$ of some center.
 - Thus, within each phase, every $q \in P$ is scanned at most $I^{O(1)}$ times.
 - Since there are at most $\log F$ phases, the scanning takes $I^{O(1)} n \log F$ time.
- The max-heap can be maintained in $I^{O(1)} n \log n$ time.
- Updating the friends lists take $I^{O(1)} n$ time in total.

Greedy Clustering: Runtime

- Theorem: Let P be an n -point metric space with doubling constant I . Then, a greedy permutation of P can be computed in $O(I^{O(1)} n \log(Fn))$ time and $O(I^{O(1)} n)$ space.
- Question: Can we remove the dependence on spread?

A Special Case: Ultrametrics

- Suppose that the underlying metric is an ultrametric, i.e. for all $x, y, z \in \mathcal{P}$, we have $d(x, z) \leq \max\{d(x, y), d(y, z)\}$.
- It is known that the class of ultrametrics coincides with the class of *hierarchical well-separated trees* (HST).

A Special Case: Ultrametrics

- Definition: An HST is a metric space defined on the leaves of a rooted tree T . Every $u \in T$ has a label $D_u \geq 0$ s.t. $D_u = 0$ iff u is a leaf. If u is a child of v , then $D_u \leq D_v$. The distance between two leaves is $d_{u,v}$ defined by $D_{lca(u,v)}$.
 - We assume WLOG that T is binary.
 - Each node $u \in T$ has a representative rep_u which is an arbitrary leaf of the subtree rooted at u . As before, we enforce an inheritance property.

Greedy Clustering Revisited

- Idea: Use the tree structure of T to help us find “wide” separating rings and guide our greedy algorithm.
- We apply the greedy clustering algorithm to a *dynamic* point set that will correspond to a level of the HST.
 - As the algorithm progresses, the point set will grow until it includes the entire set P .

Greedy Clustering Revisited

- Suppose we have a max-heap H that maintains the nodes of the HST sorted by the D s.
- Initially, the dynamic point set consists of $\{p_1, rep(u_r)\}$, where p_1 is an arbitrary center, and u_r is the root of the HST.
- We also keep a max-heap H' that maintains the distances of the points in the current set to their centers.

Greedy Clustering Revisited

- Let r_{cur} be the current max cluster radius, u be the result of an extraction from H .
- If $D_u \leq r_{cur} / n^2 \leq D_{p(u)}$, then we proceed as in the greedy algorithm.
- O/W, we replace u by its children v, w .
 - (i) Add $\{rep_v, rep_w\} \setminus \{rep_u\}$ to the cluster p_i if rep_u belongs to p_i .
 - (ii) Update the nearest centers by scanning the friends list of p_i .
 - (iii) Insert $\{rep_v, rep_w\} \setminus \{rep_u\}$ to H' .

Greedy Clustering Revisited

- Let $r_k = \min_{1 \leq i \leq k} d(p_{k+1}, p_i)$. We observe that:

$$P \subset \bigcup_{i=1}^k B(p_i, (1+n^{-2})r_k)$$

- In particular, this shows that it suffices to scan the friends list of p_i in Step (ii).
- How about the runtime?
 - As before, we divide the execution of the algorithm into phases.
 - In the i -th phase, the algorithm handles clusters with radii in the range

$$[\text{diam}(P)2^{-(i+1)}, \text{diam}(P)2^{-i}]$$

Greedy Clustering Revisited

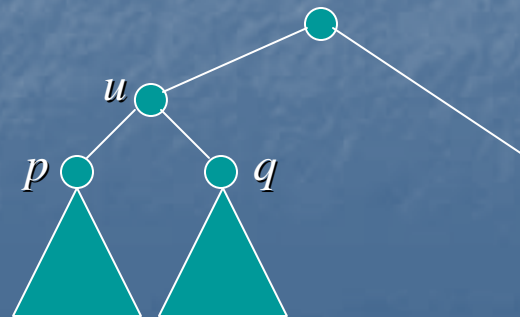
- Consider a point $p \in \mathcal{P}$: it is inserted when a node $u \in \mathcal{T}$ is split. Let p, q be the representatives of the 2 children. We charge u for the work done on p, q in the next $L = 6 \log n$ phases.
- Now, consider the work done on p before it undergoes another split event.
- If p is at most L phases away from its split event at u , then u pays for it.

Greedy Clustering Revisited

- Thus, consider p at $>L$ phases away from its split event at u . Note that:
 - p represents a point set of diam. $\leq r_{cur}/n^2$;
 - $r_{cur} \leq D_u/n^4$.
- These imply that:

$$P \cap B(p, r_{cur} \cdot n^2) \subset P \cap B\left(p, \frac{\Delta_u}{n^2}\right) \subset P \cap B\left(p, \frac{r_{cur}}{n^2}\right)$$

where the second inclusion follows from the structure of the HST.



Greedy Clustering Revisited

- Thus, all the points represented by p is far away from the rest of P in terms of r_{cur} .
 - Intuitively, there is a wide separating ring built into the structure of the HST wrt r_{cur} .
 - In particular, the friends lists don't interfere!
 - p does not require any work.
- Hence, every node in the HST is charged with $I^{O(1)} \log n$ work, which yields a total runtime of $O(I^{O(1)} n \log n)$.

Greedy Clustering Revisited

- For the general case, we need to embed (fast) our metric into an HST (which incurs distortion).
- Sketch: This involves two steps.
 - Build a sparse (roughly $O(l^{O(1)} n \log n)$ edges) spanner on the underlying metric.
 - Embed the spanner into an HST.
- The first step is needed because the runtime of the second step depends linearly on the number of edges.

Construction of Net-Tree

- The tree constructed for p_1, \dots, p_k will be denoted by $T^{(k)}$. We want $T = T^{(n)}$.
 - In particular, we build T inductively.
- We maintain for each $u \in T^{(k)}$ a set of "close-by" vertices $Rel(u)$:

$$Rel(u) = \{v \in T^{(k)} : l(v) \leq l(u) \leq l(p(v)), d(rep_u, rep_v) \leq 13b^{l(u)}\}$$

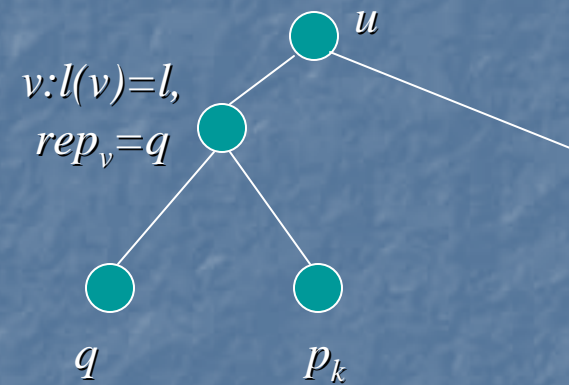
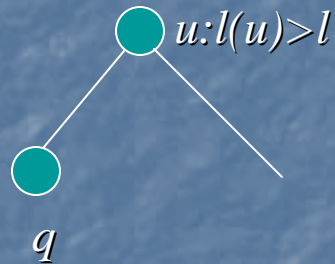
Construction of Net-Tree

- Let $r_i' = \min\{r_j : 1 \leq j \leq i\}$.
- The k -th point p_k is added as a leaf, and set the parameters accordingly.
- Let $l = \lceil \log_b r_{k-1}' \rceil$ and h be the largest index s.t. $\lceil \log_b r_{h-1}' \rceil > l$.
- Let $q \in \{p_1, \dots, p_h\}$ be the closest point to p_k . Consider the leaf of $T^{(k-1)}$ corresponding to q , and let $u = p(q)$.

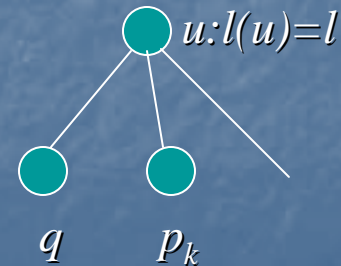
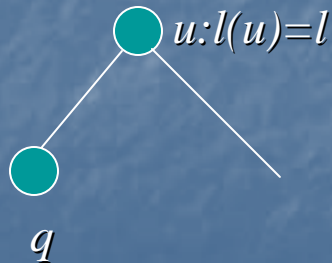
Construction of Net-Tree

- We get $\mathcal{T}^{(k)}$ as follows.

- Case 1: $l(u) > l$



- Case 2: $l(u) = l$



Construction of Net-Tree

- The main algorithmic tasks are:
 - find q_i ;
 - update the sets $Rel(\cdot)$.