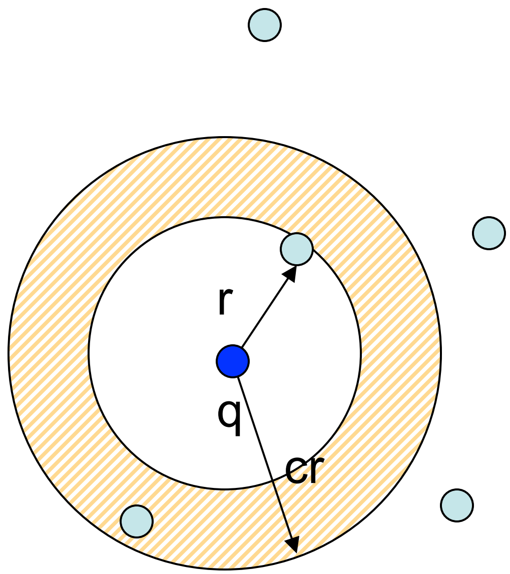
Similarity Search in High Dimensions II

Piotr Indyk

MIT

Approximate Near(est) Neighbor

- c-Approximate Nearest Neighbor: build data structure which, for any query q
  - returns $p' \in P$, $||p-q|| \leq cr$,
  - where r is the distance to the nearest neighbor of q
- c-Approximate r-Near Neighbor: build data structure which, for any query q:
  - If there is a point $p \in P$, $||p-q|| \leq r$
  - it returns $p' \in P$, $||p-q|| \leq cr$



Algorithms for c-Approximate Near Neighbor

Space	Time	Comment	Norm	Ref
$dn+n^{O(1/\varepsilon^2)}$	$d * \log n / \varepsilon^2$ (or 1)	$c=1+ \varepsilon$	Hamm, l_2	[KOR'98, IM'98]
$n^{\Omega(1/\varepsilon^2)}$	$O(1)$			[AIP'06]
$dn+n^{1+\rho(c)}$	$dn^{\rho(c)}$	$\rho(c)=1/c$	Hamm, l_2	[IM'98], [GIM'98],[Cha'02]
		$\rho(c)<1/c$	l_2	[DIIM'04]
$dn * \log s$	$dn^{\sigma(c)}$	$\sigma(c)=O(\log c/c)$	Hamm, l_2	[Ind'01]
$dn+n^{1+\rho(c)}$	$dn^{\rho(c)}$	$\rho(c)=1/c^2 + o(1)$	l_2	[AI'06]
		$\sigma(c)=O(1/c)$	l_2	[Pan'06]

Reductions

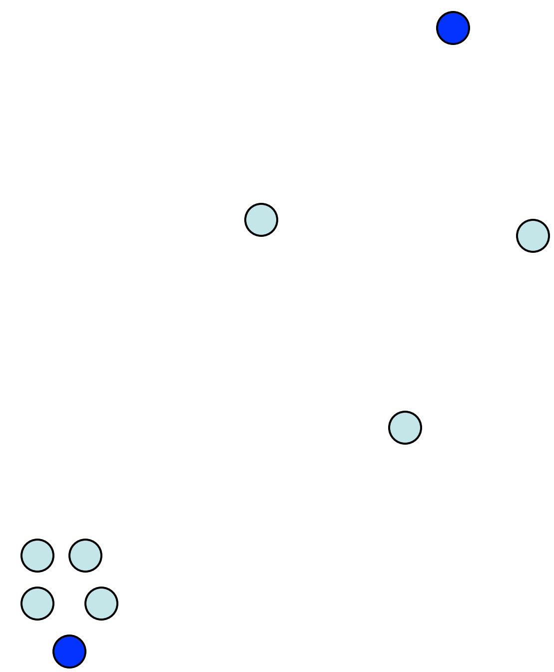
- $c(1+\gamma)$-Approx Nearest Neighbor reduces to c-Approx Near Neighbor
- Easy:
  - Space multiplied by $(\log \Delta)/\gamma$, where Δ is the spread, i.e., all distances in P are in $[1 \ldots \Delta]$
  - Query time multiplied by $\log((\log \Delta)/\gamma)$
  - Probability of failure multiplied by $(\log \Delta)/\gamma$
  - Idea:
    - Build data structures with $r = \frac{1}{2}, \frac{1}{2}(1+\gamma), \frac{1}{2}(1+\gamma)^2, \ldots, O(\Delta)$
    - To answer query, do binary search on values of r
- Hard: replace $\log \Delta$ by $\log n$

General reduction
[Har Peled-Indyk-Motwani'11]

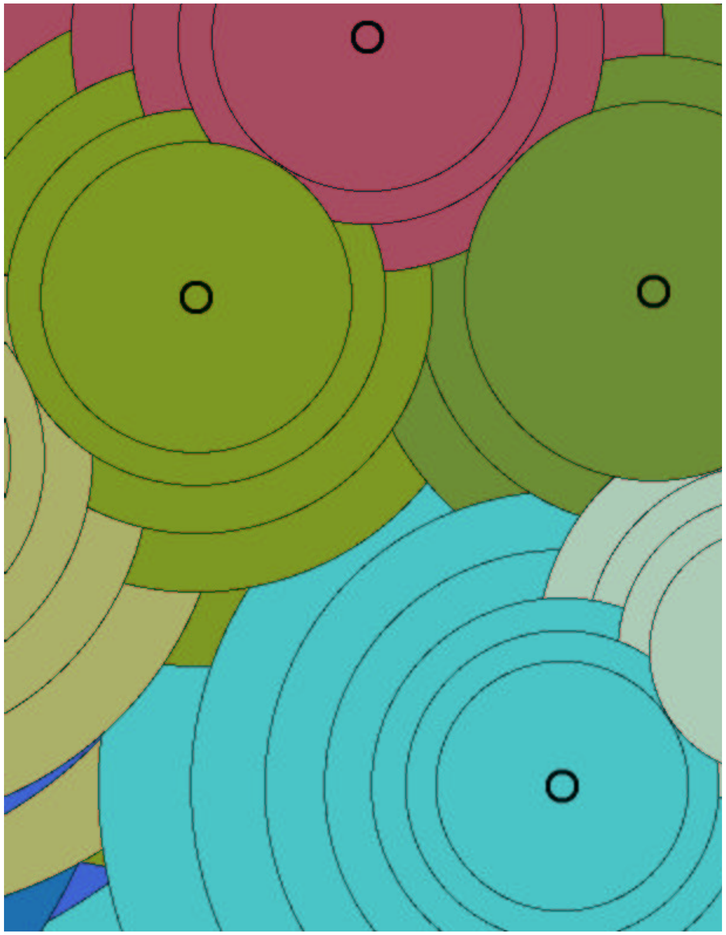
- Assume we have a data structure for dynamic c-Near Neighbor in under a metric D which, for parameters n,f has:
  - $T(n,c,f)$ construction time
  - $S(n,c,f)$ space bound
  - $Q(n,c,f)$ query time
  - $U(n,c,f)$ update time
- Then we get a data structure for $c(1+O(\gamma))$-Nearest Neighbor with:
  - $O(T(n,c,f)/\gamma \cdot \log^2 n + n\log n[Q(n,c,f)+U(n,c,f)])$ expected construction time
  - $O(S(n,c,f)/\gamma \cdot \log^2 n)$ space bound
  - $Q(n,c,f)\ O(\log n)$ query time
  - $O(f \log n)$ failure probability
- Generalizes, improves, simplifies and merges [Indyk-Motwani'98] and [Har Peled'01]

Intro

- Basic idea: use different scales (i.e., radiuses r) for different clouds of points
  - At most n^2 total
  - Would like $(\log n)^2$ per point, on the average
- We will see a simplified reduction:
  - From approximate nearest neighbor to exact near neighbor
  - Simplifying assumption
- Actual reduction a little more complex, but follows the same idea

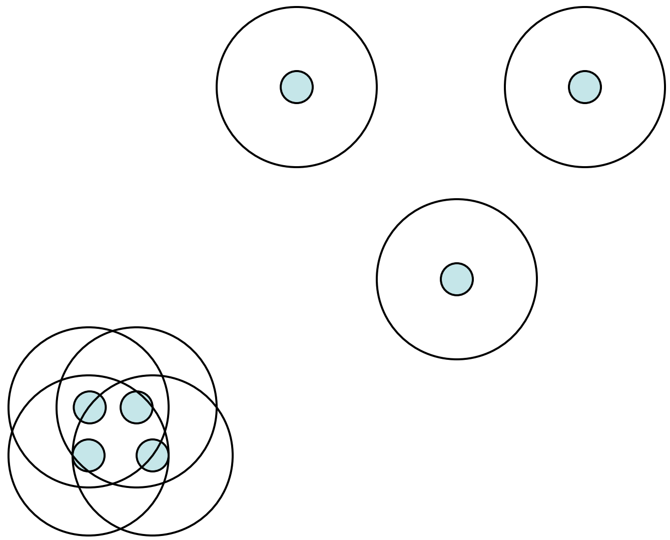


Example



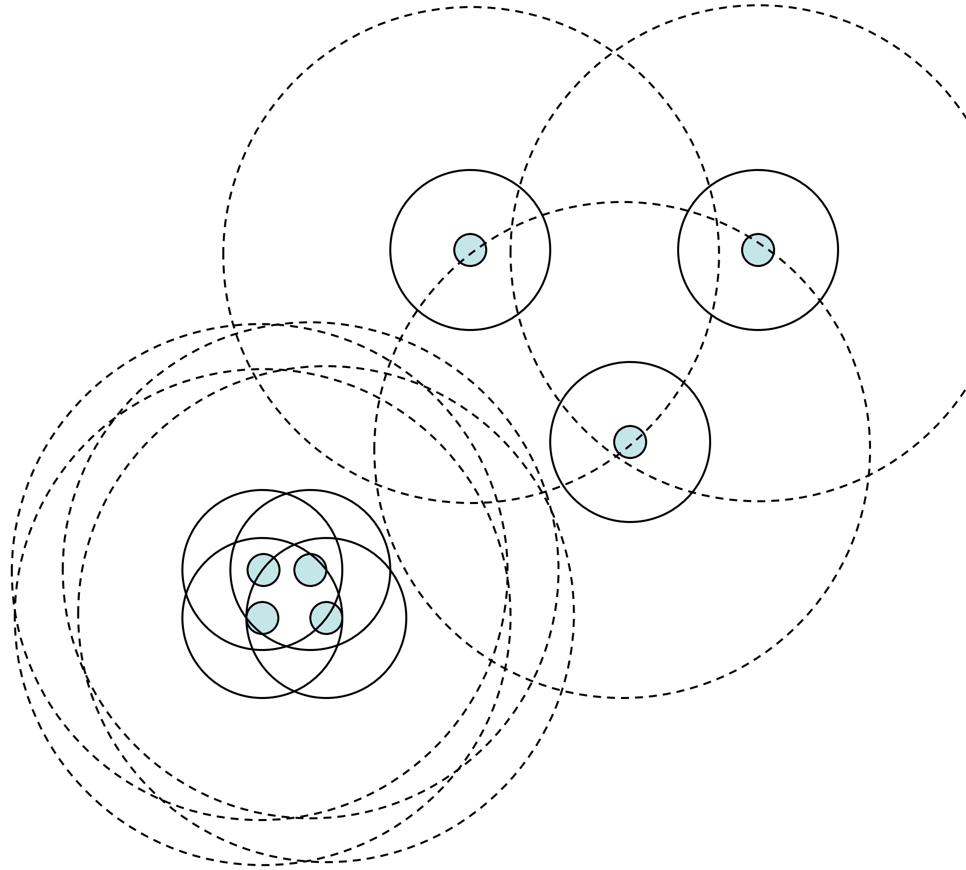
Notation

- $UB_P\ (r) = \cup_{p \in P} B(p, r)$
- $CC_P(r)$ is a partitioning of P induced by the connected components of $UB_P(r)$
- r_{med} is the smallest value of r such that $UB_P(r)$ has a component of size at least $n/2 + 1$
- $UB_{med}=UB_P(r_{med})$
- $CC_{med}=CC_P(r_{med})$
- Simplifying assumption: $UB_P(r_{med})$ has a component of size exactly $n/2+1$



A simplified reduction

- Set $r_{top} = \Theta(n r_{med} \log(n)/\gamma)$
- Exact near neighbor data structures NN:
  - For $i=0\ldots\log_{1+\gamma}(2r_{top}/r_{med})$,
    create $NN(P, r_{med}(1+\gamma)^i/2)$
  - For each component $C \in CC_{med}$ recurse on C
  - Recurse on $P' \subset P$ that contains one point per each component $C \in CC_{med}$ (at most $n/2$ points)
- Note that the recursion has depth $O(\log n)$



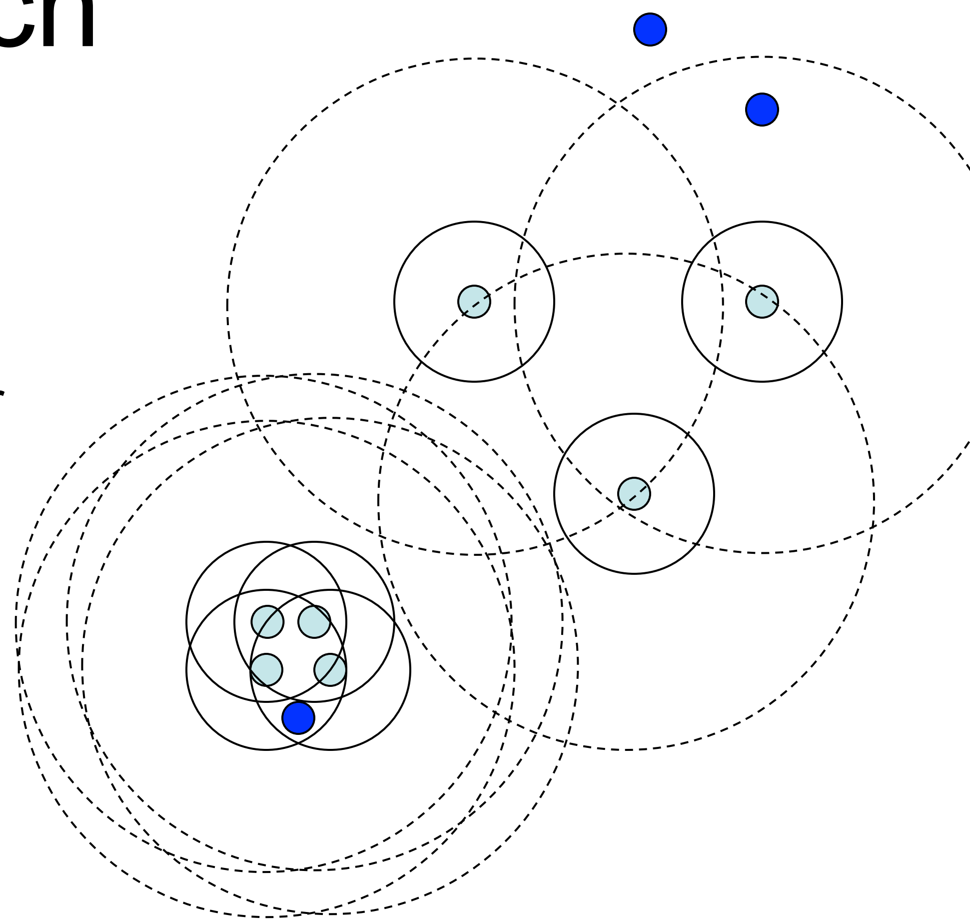
Inner radius $=r_{med}/2$

Outer radius $=r_{top}$

Search

1. Use $NN(P, r_{med}/2)$ to check whether $D(q,P)<r_{med}/2$
   - If yes, recurse on the component C containing q
2. Else use $NN(P, r_{top})$ to check whether $D(q,P)>r_{top}$
   - If yes, recurse on P'
3. Else perform binary search on

   $NN(P, r_{med}(1+\gamma)^i/2)$



- Correctness for Cases 1 and 3 follows from the procedure
- Case 2 need a little work:
  – Each "contraction" introduces an additive error up to $n\ r_{med}$
  – But the distance to nearest neighbor lower-bounded by $r_{top} = \Theta(nr_{med} \log(n)/\gamma)$
  – Accumulated relative error at most $(1+n\ r_{med}/r_{top})^{O(\log n)} = (1+\gamma/\log(n))^{O(\log n)}$

Space

- Let $B(n)$ be the maximum number of points stored by the data structure
  - Space = $O(B(n) \log_{1+\gamma} (r_{top}/r_{med}))$
- We have

  $$B(n)= \max_{k,n1+n2+\ldots+nk=n} \Sigma_i B(n_i)+B(k)+n$$

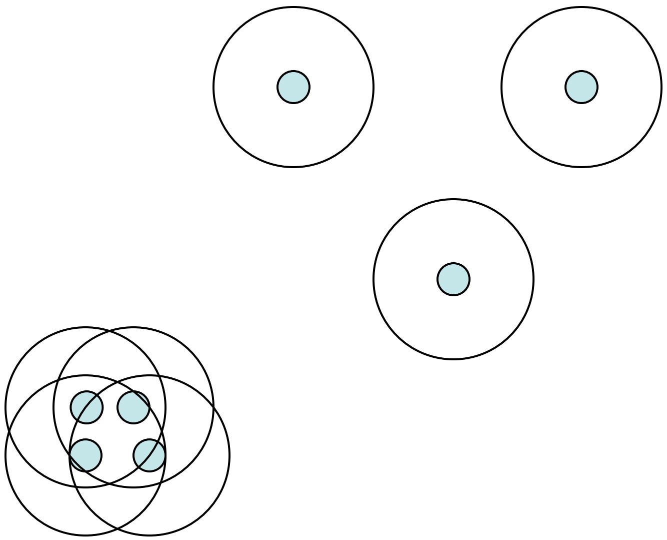
  subject to $k \le n/2$,$1 \le n_i \le n/2$
- This solves to $O(n \log n)$

Construction time

- Estimating r_{med}
  - Selects a point p uniformly at random from P
  - Return r^* =median of the set $D(p,p')$ over $p' \in P$
  - We have

    $$r_{med} \leq r^* \leq (n-1) r_{med}$$

    with probability >1/2
- Approximating $CC_P(r^*)$
  - n queries and updates to NN with r^*



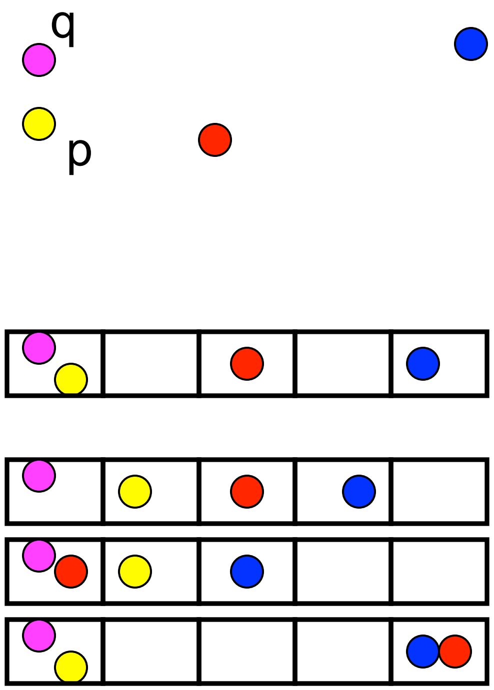
Algorithms for c-Approximate Near Neighbor

Space	Time	Comment	Norm	Ref
$dn+n^{O(1/\varepsilon^2)}$	$d * \log n / \varepsilon^2$ (or 1)	$c=1+\varepsilon$	Hamm, l_2	[KOR'98, IM'98]
$n^{\Omega(1/\varepsilon^2)}$	$O(1)$			[AIP'06]
$dn+n^{1+\rho(c)}$	$dn^{\rho(c)}$	$\rho(c)=1/c$	Hamm, l_2	[IM'98], [GIM'98],[Cha'02]
		$\rho(c)<1/c$	l_2	[DIIM'04]
$dn * \log s$	$dn^{\sigma(c)}$	$\sigma(c)=O(\log c/c)$	Hamm, l_2	[Ind'01]
$dn+n^{1+\rho(c)}$	$dn^{\rho(c)}$	$\rho(c)=1/c^2 + o(1)$	l_2	[AI'06]
		$\sigma(c)=O(1/c)$	l_2	[Pan'06]

Locality-Sensitive Hashing

[Indyk-Motwani'98]

- Idea: construct hash functions $g: R^d \rightarrow U$ such that for any points p,q:
  - If $||p-q|| \le r$, then $Pr[g(p)=g(q)]$ is ~~"high"~~ "not-so-small"
  - If $||p-q|| > cr$, then $Pr[g(p)=g(q)]$ is "small"
- Then we can solve the problem by hashing
- Related work: [Paturi-Rajasekaran-Reif'95, Greene-Parnas-Yao'94, Karp-Waarts-Zweig'95, Califano-Rigoutsos'93, Broder'97]



LSH

- A family H of functions $h: R^d \to U$ is called (P_1,P_2,r,cr)-sensitive, if for any p,q:
  - if $||p-q|| < r$ then $Pr[\ h(p)=h(q)\ ] > P_1$
  - if $||p-q|| > cr$ then $Pr[\ h(p)=h(q)\ ] < P_2$

- Example: Hamming distance
  - KOR'98: $h(p) = \Sigma_{i \in S}\ p_i\ u_i \bmod 2$
  - IM'98: $h(p)=p_i$, i.e., the i-th bit of p
    - Probabilities: $Pr[\ h(p)=h(q)\ ] = 1-H(p,q)/d$

p=10010010
q=11010110

Algorithm

- We use functions of the form

  $$g(p)=<h_1(p),h_2(p),\ldots,h_k(p)>$$

- Preprocessing:
  - Select $g_1 \ldots g_L$
  - For all $p \in P$, hash p to buckets $g_1(p) \ldots g_L(p)$

- Query:
  - Retrieve the points from buckets $g_1(q)$, $g_2(q)$, … , until
    - Either the points from all L buckets have been retrieved, or
    - Total number of points retrieved exceeds $3L$
  - Answer the query based on the retrieved points
  - Total time: $O(dL)$

Analysis [IM'98, Gionis-Indyk-Motwani'99]

- Lemma1: the algorithm solves c-approximate NN with:
  - Number of hash functions:

    $$L=n^{\rho},\ \rho=\log(1/P1)/\log(1/P2)$$
  - Constant success probability per query q
- Lemma 2: for Hamming LSH functions, we have $\rho=1/c$

Proof of Lemma 1 by picture

- Points in $\{0,1\}^d$
- Collision prob. for k=1..3, L=1..3 (recall: L=#indices, k=#h's)
- Distance ranges from 0 to d=10