cse 521: design and analysis of algorithms

Time & place
T, Th 1200-120 pm in CSE 203

People
Prof: James Lee (jrl@cs)
TA: Thach Nguyen (ncthach@cs)

Book
Algorithm Design by Kleinberg and Tardos

Grading
50% homework (approx. bi-weekly problem sets)
20% take-home midterm
30% in-class final exam

Website: http://www.cs.washington.edu/521/
something a little bit different

assume you know:
  asymptotic analysis
  basic probability
  basic linear algebra
  dynamic programming
  recursion / divide-and-conquer
  graph traversal (BFS, DFS, shortest paths)

so that we can cover:
  nearest-neighbor search
  spectral algorithms (e.g. pagerank)
  online algorithms (multiplicative update)
  geometric hashing
  + graph algorithms, data structures, network flow, hashing, NP-completeness,
  linear programming, approx. algorithms

due: april 9th
case study: nearest-neighbor search

subset of objects forms the database

Goal:
Quickly respond with the database object most similar to the query.

universe of objects
formal model

subset of objects forms the database

Goal:
Quickly respond with the database object most similar to the query.

\[ U = \text{universe (set of objects)} \]
\[ d(x,y) = \text{distance between two objects} \]

Assumptions:
\[ d(x,x) = 0 \quad \text{for all } x \in U \]
\[ d(x,y) = d(y,x) \quad \text{for all } x,y \in U \]  
\[ \text{(symmetry)} \]
\[ d(x,y) \leq d(x,z) + d(z,y) \quad \text{for all } x,y,z \in U \]  
\[ \text{(triangle inequality)} \]

Problem:
Given an input database \( D \subseteq U \):
preprocess \( D \) (fast, space efficiently) so that queries \( q \in U \) can be answered very quickly, i.e. return \( a^* \in D \) such that \( d(q,a^*) = \min \{ d(q,x) : x \in D \} \)
Problem:
Given an input database $D \subseteq U$:
preprocess $D$ (fast, space efficiently) so that queries $q \in U$ can be answered very quickly, i.e. return
$a^* \in D$ such that $d(q, a^*) = \min \{ d(q, x) : x \in D \}$
primitive methods

[Brute force: Time]
Compute $d(\text{query}, x)$ for every object $x \in D$, and return the closest.

Takes time $\approx |D| \cdot \text{(distance comp. time)}$

[Brute force: Space]
Pre-compute best response to every possible query $q \in U$.

Takes space $\approx |U| \cdot \text{(object size)}$

Dream performance:

Query time: $O(\log |D|)$
Space: $O(|D|)$
something hard, something easy
something hard, something easy
All pairwise distances are equal:
\[ d(x,y) = 1 \quad \text{for all } x,y \in D \]

Problem:
... so that queries \( q \in U \) can be answered quickly, i.e. return \( a^* \in D \) such that
\[ d(q,a^*) = \min \{ d(q,x) : x \in D \} \]
something hard, something easy

All pairwise distances are equal:
\[ d(x,y) = 1 \text{ for all } x,y \in D \]

\[ \epsilon \text{-Problem:} \]

... so that queries \( q \in U \) can be answered quickly, i.e. return an \( a \in D \) such that \( d(q,a) \leq (1+\epsilon) d(q,D) \)
Let’s suppose that \( U = [0,1] \) (real numbers between 0 and 1).

Answer: Sort the points \( D \subseteq U \) in the preprocessing stage.
To answer a query \( q \in U \), we can just do binary search.
To support insertions/deletions in \( O(\log |D|) \) time, can use a BST.
(balanced search tree)

How much power did we need?
Can we do this just using distance computations \( d(x,y) \)? (for \( x,y \in D \))

Basic idea: Make progress by throwing “a lot” of stuff away.
**Definition:** The ball of radius $\alpha$ around $x \in D$ is

$$B(x, \alpha) = \{ y \in D : d(x, y) \leq \alpha \}$$
Definition: The ball of radius $\alpha$ around $x \in D$ is

$$B(x, \alpha) = \{y \in D : d(x, y) \leq \alpha\}$$
extending the basic idea

**Definition:** An $\alpha$-net in $D$ is a subset $N \subseteq D$ such that

1) **Separation:** For all $x, y \in N$, $d(x, y) \geq \alpha$
2) **Covering:** For all $x \in D$, $d(x, N) \leq \alpha$

**Greedy construction algorithm:**

Start with $N = \emptyset$.
As long as there exists an $x \in D$ with $d(x, N) > \alpha$, add $x$ to $N$.

So for every $\alpha > 0$, we can construct an $\alpha$-net $N(\alpha)$ in $O(n^2)$ time, where $n = |D|$.
basic data structure: hierarchical nets

Search algorithm:
- Use the $\alpha$-net to find a radius $2\alpha$ ball.
- Use the $\alpha/2$-net to find a radius $\alpha$ ball.
- Use the $\alpha/4$-net to find a radius $\alpha/2$ ball.
**Data structure:**

\[d_{\text{max}} = \max \{d(x, y) : x, y \in D\}\]

\[d_{\text{min}} = \min \{d(x, y) : x \neq y \in D\}\]

For \(i = \log(d_{\text{min}}), \log(d_{\text{min}}) + 1, \ldots, \log(d_{\text{max}})\), let \(N_i\) be a \(2^i\)-net.  

For each \(x \in N_i\), \(L_{x,i} = B(x, 2^{i+1}) \cap N_{i-1}\).  

[Diagram of hierarchical nets]
Data structure:

\[ d_{\text{max}} = \max \{d(x, y) : x, y \in D\} \]
\[ d_{\text{min}} = \min \{d(x, y) : x \neq y \in D\} \]

For \( i = \log(d_{\text{min}}), \log(d_{\text{min}}) + 1, \ldots, \log(d_{\text{max}}) \),

let \( N_i \) be a \( 2^i \)-net.

For each \( x \in N_i \), \( L_{x,i} = B(x, 2^{i+1}) \cap N_{i-1} \).

Algorithm: Given input query \( q \in U \),

Let \( \text{CurrentPoint} = \) only point of \( N_{\log(d_{\text{max}})} \).

For \( i = \log(d_{\text{max}}) - 1, \log(d_{\text{max}}) - 2, \ldots, \log(d_{\text{min}}) \),

\( \text{CurrentPoint} = \) closest point to \( q \) in \( L_{\text{CurrentPoint},i} \).
Algorithm: Given input query \( q \in U \),

Let \( \text{CurrentPoint} = \) only point of \( N_{\log(d_{\text{max}})} \).

For \( i = \log(d_{\text{max}}) - 1, \log(d_{\text{max}}) - 2, \ldots, \log(d_{\text{min}}) \),

\( \text{CurrentPoint} = \) closest point to \( q \) in \( L_{\text{CurrentPoint},i} \)
Query time = $O\left(\log \left(\frac{d_{\text{max}}}{d_{\text{min}}}\right)\right) \max \left\{|L_{x,i}| : x \in D, i\right\}$

$L_{x,i} = B(x, 2^{i+1}) \cap N_{i-1}$

**Nearly uniform point set:**
For $u,v \in L_{x,i}$, $d(u,v) \in [2^{i-1}, 2^{i+2}]$

**Algorithm:** Given input query $q \in U$,

- Let CurrentPoint = only point of $N_{\log(d_{\text{max}})}$.
- For $i = \log(d_{\text{max}}) - 1, \log(d_{\text{max}}) - 2, \ldots, \log(d_{\text{min}})$,
  CurrentPoint = closest point to $q$ in $L_{\text{CurrentPoint},i}$
curs’ed hamsters

All pairwise distances are equal:

\[ d(x,y) = 1 \quad \text{for all } x, y \in D \]
Given a metric space \((X,d)\), let \(\lambda(X,d)\) be the smallest constant \(\lambda\) such that every ball in \(X\) can be covered by \(\lambda\) balls of half the radius.

The \textbf{intrinsic dimension} of \((X,d)\) is the value

\[
\dim(X, d) = \log_2 \lambda(X, d)
\]
We can bound the query time of our algorithm in terms of the **intrinsic dimension** of the data...

**Query time** = \( O \left( \log \left( \frac{d_{\max}}{d_{\min}} \right) \right) \max \left\{ |L_{x,i}| : x \in D, i \right\} \)

\[ L_{x,i} = B(x, 2^{i+1}) \cap N_{i-1} \]

**Claim:** \( |L_{x,i}| \leq [\lambda(X,d)]^3 \)

**Proof:** Suppose that \( k = |L_{x,i}| \). Then we need at least \( k \) balls of radius \( 2^{i-2} \) to cover \( B(x, 2^{i+1}) \), because a ball of radius \( 2^{i-2} \) can cover at most one point of \( N_{i-1} \).

But now we claim that (for any \( r \)) every ball of radius \( r \) in \( X \) can be covered by at most \([\lambda(X,d)]^3\) balls of radius \( r/8 \), hence \( k \leq [\lambda(X,d)]^3 \).
But now we claim that (for any $r$) every ball of radius $r$ in $X$ can be covered by at most $[\lambda(X,d)]^3$ balls of radius $r/8$, hence $k \leq [\lambda(X,d)]^3$.

A ball of radius $r$ can be covered $\lambda$ balls of radius $r/2$, hence by $\lambda^2$ balls of radius $r/4$, hence by $\lambda^3$ balls of radius $r/8$. 
We can bound the query time of our algorithm in terms of the intrinsic dimension of the data...

$$\text{Query time} = O \left( \log \left( \frac{d_{\text{max}}}{d_{\text{min}}} \right) \right) \max \left\{ |L_{x,i}| : x \in D, i \right\}$$

$$L_{x,i} = B(x, 2^{i+1}) \cap N_{i-1}$$

Claim: $$|L_{x,i}| \leq [\lambda(X,d)]^3$$

Proof: Suppose that $$k = |L_{x,i}|$$. Then we need at least $$k$$ balls of radius $$2^{i-2}$$ to cover $$B(x, 2^{i+1})$$, because a ball of radius $$2^{i-2}$$ can cover at most one point of $$N_{i-1}$$.

But now we claim that (for any $$r$$) every ball of radius $$r$$ in $$X$$ can be covered by at most $$[\lambda(X,d)]^3$$ balls of radius $$r/8$$, hence $$k \leq [\lambda(X,d)]^3$$. 
We can bound the query time of our algorithm in terms of the intrinsic dimension of the data…

\[
\text{Query time} = O \left( \log \left( \frac{d_{\text{max}}}{d_{\text{min}}} \right) \right) [\lambda(X, d)]^3
\]

\[
L_{x,i} = B(x, 2^{i+1}) \cap N_{i-1}
\]

**Claim:** \( |L_{x,i}| \leq [\lambda(X,d)]^3 \)

**Proof:** Suppose that \( k = |L_{x,i}| \). Then we need at least \( k \) balls of radius \( 2^{i-2} \) to cover \( B(x, 2^{i+1}) \), because a ball of radius \( 2^{i-2} \) can cover at most one point of \( N_{i-1} \).

But now we claim that (for any \( r \)) every ball of radius \( r \) in \( X \) can be covered by at most \( [\lambda(X,d)]^3 \) balls of radius \( r/8 \), hence \( k \leq [\lambda(X,d)]^3 \).
We can bound the query time of our algorithm in terms of the intrinsic dimension of the data...

$$\text{Query time} = O \left( \log \left( \frac{d_{\text{max}}}{d_{\text{min}}} \right) \right) \left[ \lambda(X, d) \right]^3$$

- Generalization of binary search (where dimension = 1)
- Works in arbitrary metric spaces with small intrinsic dimension
- Didn’t have to think in order to “index” our database
- Shows that the hardest part of nearest-neighbor search is
- Only gives approximation to the nearest neighbor
- **Next time:** Fix this; fix time, fix space + data structure prowess
- **In the future:** Opening the black box; **NNS in high dimensional spaces**
- **Bonus:** Algorithm is completely intrinsic (e.g. isomap)