Nearest Neighbor Searching in Metric Spaces

Ken Clarkson

Bell Labs

The Problem

Given a set S of n sites (points) in a metric space,
build a data structure so that:

Given a query point q, the nearest site to q can be found quickly.

Some Past Work

- Algorithms that reduce distance evals, but $\Omega(n)$ query time
  - information theory, pattern recognition literature
- *kd*-tree-like (division based on distances)
- Provable results
  - [C97] assumptions: sphere-packing bound, query distribution
  - [Karger/Ruhl] *growth-restricted* (like uniform distribution)

Here

Attempting a practical (not-necessarily-provable) algorithm.
Conditions:

- Low storage
- Never worse than brute force
- Preprocessing time/site at least as fast as query time
- Fast, even $O(\log n)$ query time, under favorable conditions
  - low-dimensional Euclidean spaces

Outline

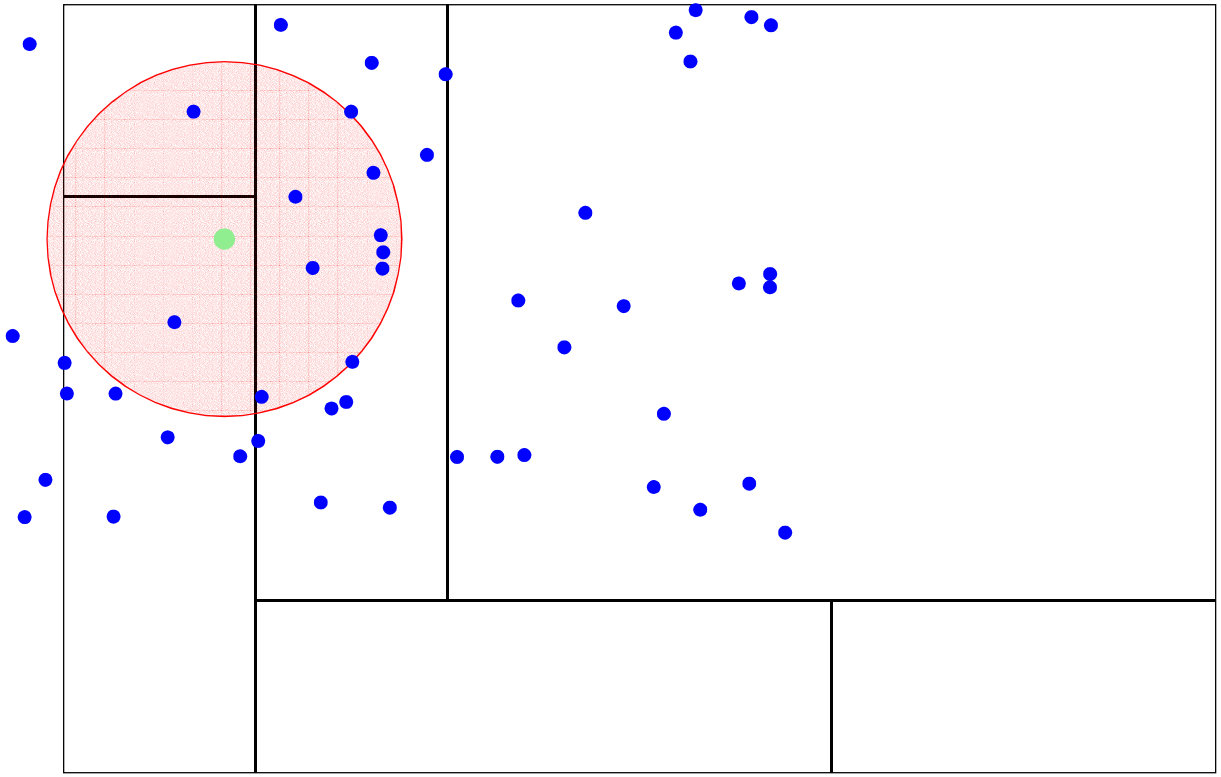
- Main ideas for the data structure
- Some experimental results
- Test harness

The main idea

- A "*kd*-tree-like" approach
- In *kd*-trees, boxes provide a simple test of too-far-away
- Use bounding spheres of Voronoi regions instead

A "*kd*-tree" approach

Only look in boxes that touch the query ball.



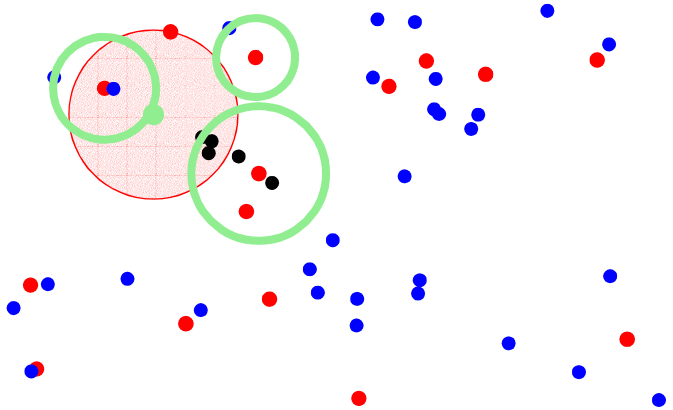
The approach here

Use a subset of the sites, called *leaders*.

- Partition by:
  - Voronoi diagram of leaders
    - Just map to nearest leader
    - Called here *Voronoi sets*
  - Instead of axis-aligned boxes
- Bounding volume is:
  - Bounding sphere of Voronoi set
    - Radius is dist to farthest in Voronoi set
  - Instead of axis-aligned boxes

The approach here

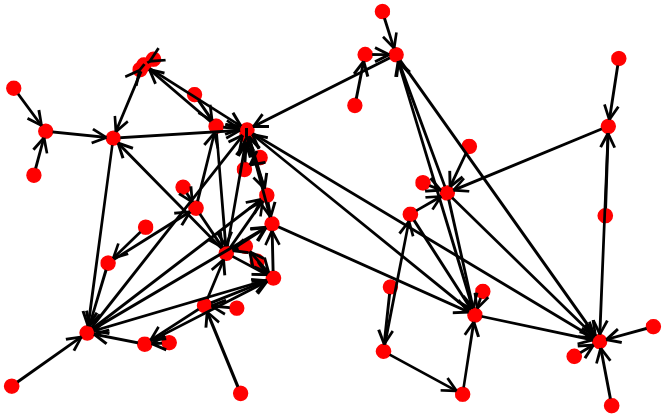
Only look in bounding balls that touch the query ball.



The data structure

- Add sites to leader set incrementally
- Maintain Voronoi sets of leaders
  - Set of sites with given leader closest
- If new leader p_i reduces the Voronoi set of leader p_j, say that p_i *touches* p_j
- Record of all "touching" relations is the data structure.
- Key process is "inverse nearest neighbor search"
  - Uses current data structure for speedup
- Insertion order that packs leaders is better
  - Next leader is site farthest from current set

Building the Data Structure



Ave arrows/leader: 1.54

Using the Data Structure

- Maintain set of *pending* leaders;
  - Initially $\{p_0\}$
  - Haven't proved that their Voronoi sets are too far away
- Consider sites touching the pending sites
  - Make them pending, if needed
  - When added, Voronoi set of pending site shrinks

Implementation

- In C
  - and C++
- Core code is about 900 lines
  - Multiple functions for answering queries
  - Fixed radius, *k*-nearest queries
  - Allows approximate queries
  - Construction stops after inserting $n/10$ sites
- Pools
  - Speed up malloc, maybe
- Heaps
- Pools, heaps, and main data structure have a "per-file modularity"

Thanks, Heaps (Ode to Floyd)

A simple, low-storage priority queue. Use for:

- Next site at max distance from leaders
  - for packing insertion
- Per leader, farthest site in Voronoi set
  - Output-sensitive 1-d range queries: find all sites in Voronoi set at distance $\geq X$, for some X.
    - For inverse searching
- In queries,
  - get the closest pending leader to query
  - In fixed-radius searching, to return answer
- Add/Delete operations while traversing for range query can be tricky.

Thanks, Heaps (Ode to Floyd)

An opaque pointer-to-struct allows the sb-structure to be hidden from user's application.

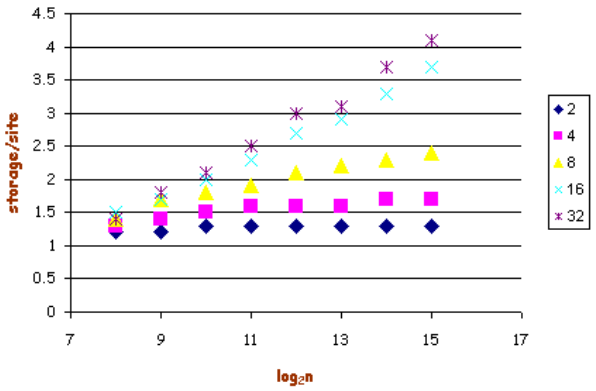
```
typedef struct sb *sbp;
sbp build_sb(
        sb_distance F,   /* pointer to distance function
        size_t num_sites
);
typedef size_t search_sb_f(
        sbp sbt,
        size_t query,   /* number of query site */
        float alpha     /* inexact search for alpha<1 */
);
```

Experiments

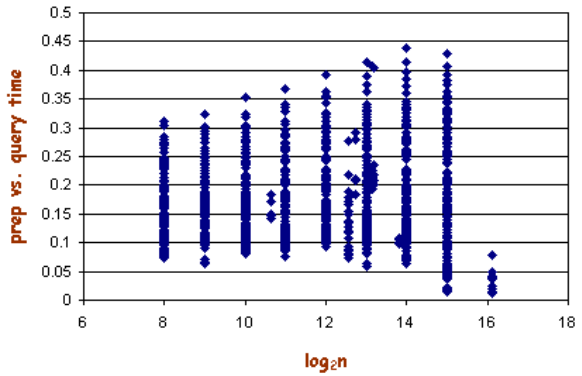
Tested for

- Euclidean data, in 2^i dimensions
  - uniform
  - normal
  - clustered normal
  - laplacian
- bit-vectors in 81d and 2304d, from OCR app
- gray-scale data in 256d, from OCR app
- strings from a wordlist; hamming and edit dist

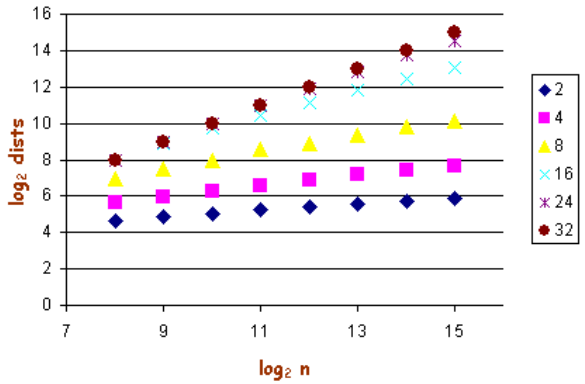
Storage, Euclidean data



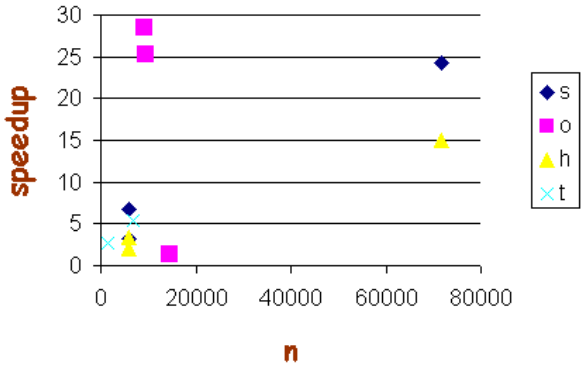
Preprocessing vs. query time, all



Query time, Euclidean



Speedup, strings and bitvectors



Test harness

Each distance measure has a:

- separate subdirectory
- `dist.c` in that subdirectory
- makefile included from parent, for building a driver
- resulting linked driver, linked with `sb` library
- record of output trials

To build the paper,

- Concatenate the trials
- Pipe all to awk script:
  - Process trials, building record of each trial
  - Append appropriate data to separate data files
  - Run metapost for figures, and laTeX to build paper

Even More Test Harness

"make distribution":

- runs all trials
- makes paper
- copies sources
- zips

Conclusions/Further Work

- Diameter?
- Accelerators
- Object-sites, using diameter bound
- Generality implies simplicity