Parallel Nearest Neighbors in Low Dimensions with Batch Updates

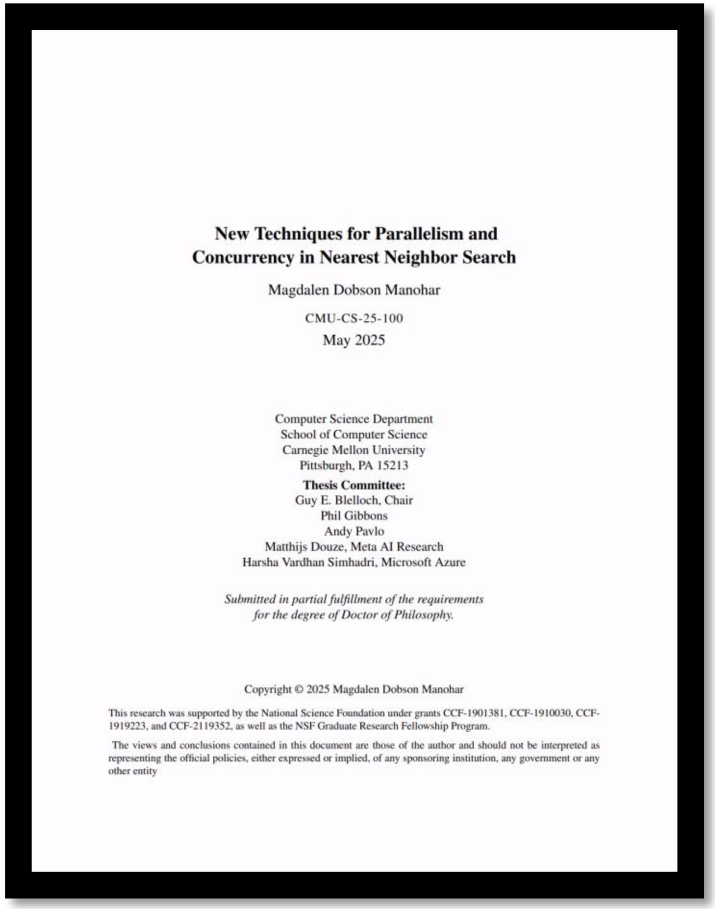
Guy E. Blelloch & Magdalen Dobson

Authors

Guy Blelloch

Magdalen Dobson

Authors



New Techniques for Parallelism and Concurrency in Nearest Neighbor Search

Magdalen Dobson Manohar

CMU-CS-25-100

May 2025

Computer Science Department
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Thesis Committee:
Guy E. Blelloch, Chair
Phil Gibbons
Andy Pavlo
Matthijs Douze, Meta AI Research
Harsha Vardhan Simhadri, Microsoft Azure

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

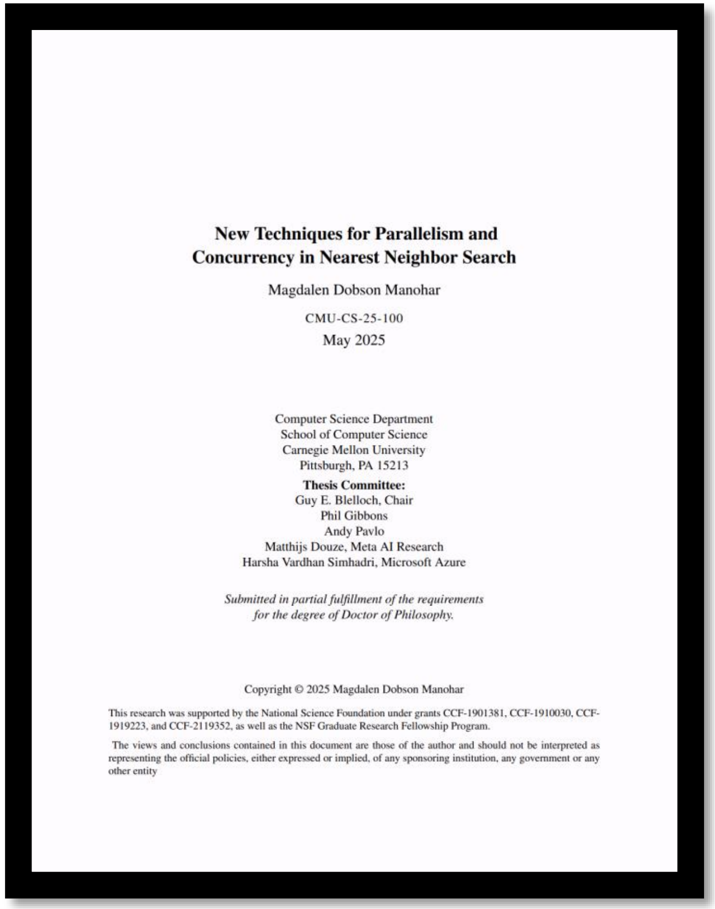
This research was supported by the National Science Foundation under grants CCF-1901381, CCF-1910030, CCF-1919223, and CCF-2119352, as well as the NSF Graduate Research Fellowship Program.

The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of any sponsoring institution, any government or any other entity

https://www.csd.cs.cmu.edu/academics/doctoral/degrees-conferred/magdalen-dobson-manohar

Magdalen Dobson

Authors



New Techniques for Parallelism and
Concurrency in Nearest Neighbor Search

Magdalen Dobson Manohar

CMU-CS-25-100
May 2025

Computer Science Department
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Thesis Committee:
Guy E. Blelloch, Chair
Phil Gibbons
Andy Pavlo
Matthijs Douze, Meta AI Research
Harsha Vardhan Simhadri, Microsoft Azure

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy.

This research was supported by the National Science Foundation under grants CCF-1901381, CCF-1910030, CCF-1919223, and CCF-2119352, as well as the NSF Graduate Research Fellowship Program.

https://www.csd.cs.cmu.edu/academics/doctoral/degrees-conferred/magdalen-dobson-manohar

- Introduces the ideas from this paper (zd-tree)
- Extends zd-tree to support concurrent queries and updates
- Modify existing high-dimensional algorithms to be lock-free, deterministic, and scalable
  - ParlayANN

Definitions

K-nearest neighbors:

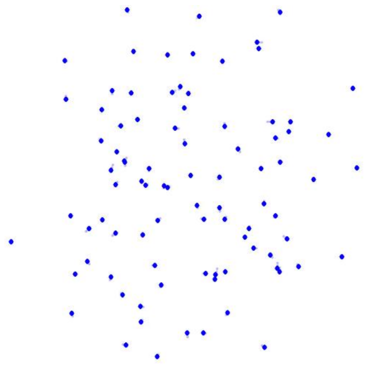
- Given a point x and set of points P, determine k nearest points to x in P

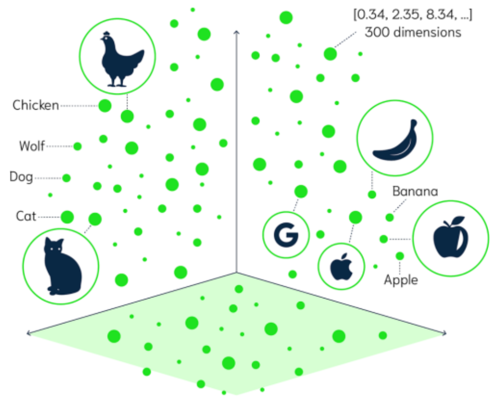
K-nearest neighbor graph:

- Calculate k-nearest-neighbors for all points in P

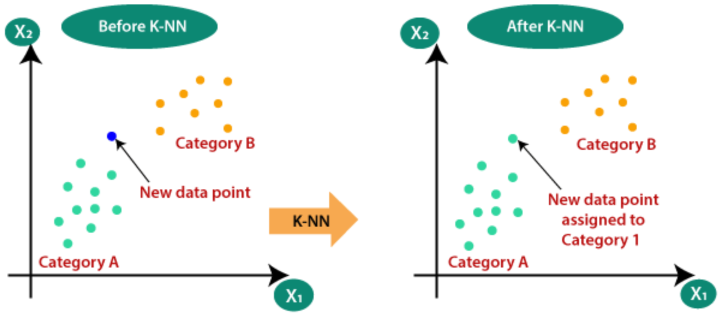
Motivation

- Nearest neighbors is a fundamental problem with many applications
  - Classification
  - Computer Graphics
  - Physics Simulations
  - Databases and IR
  - Robotics
  - Computational Biology
  - Etc.



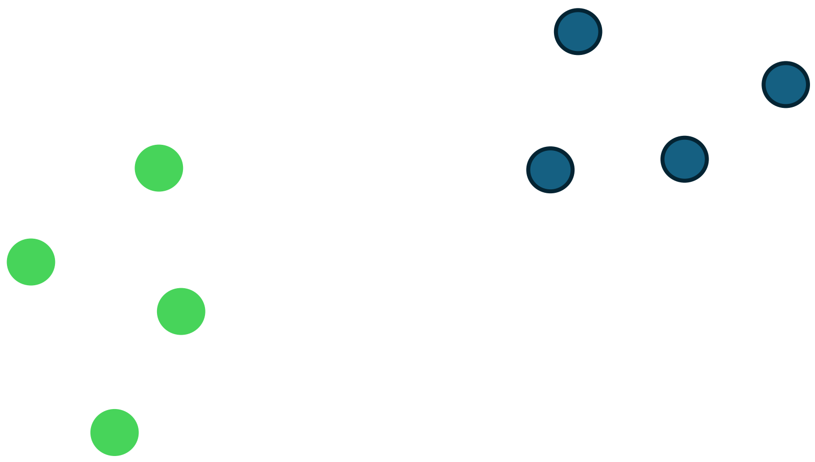


- Existing parallel implementations:
  - Aren't performant
  - Do not support batch updates



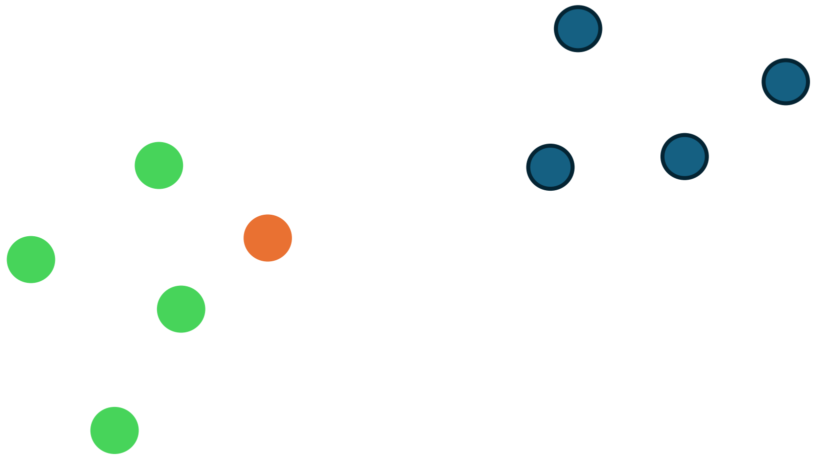
Example (Dynamic Query)

K-nearest neighbors for **new** point (k = 3)



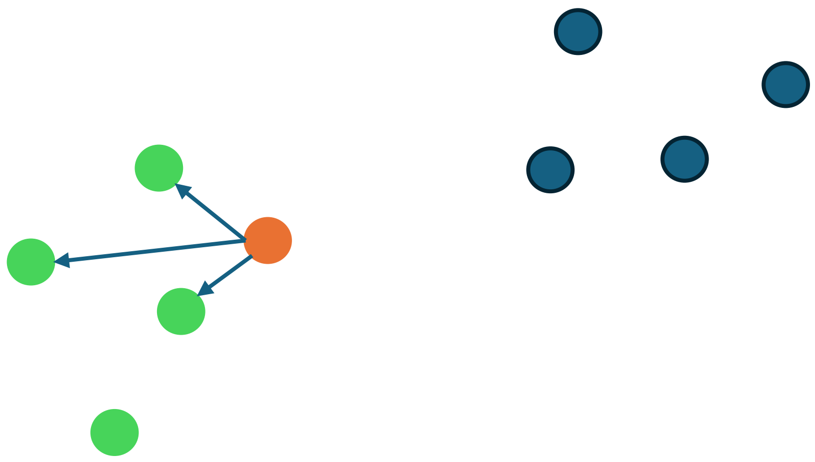
Example (Dynamic Query)

K-nearest neighbors for **new** point (k = 3)



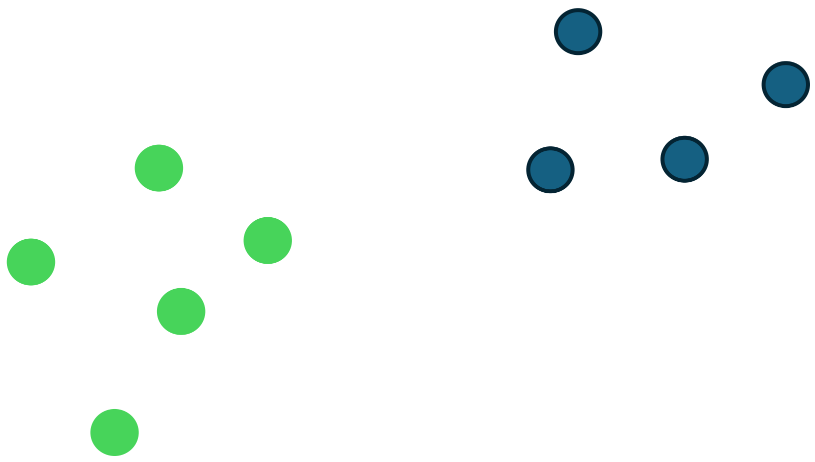
Example (Dynamic Query)

K-nearest neighbors for **new** point (k = 3)



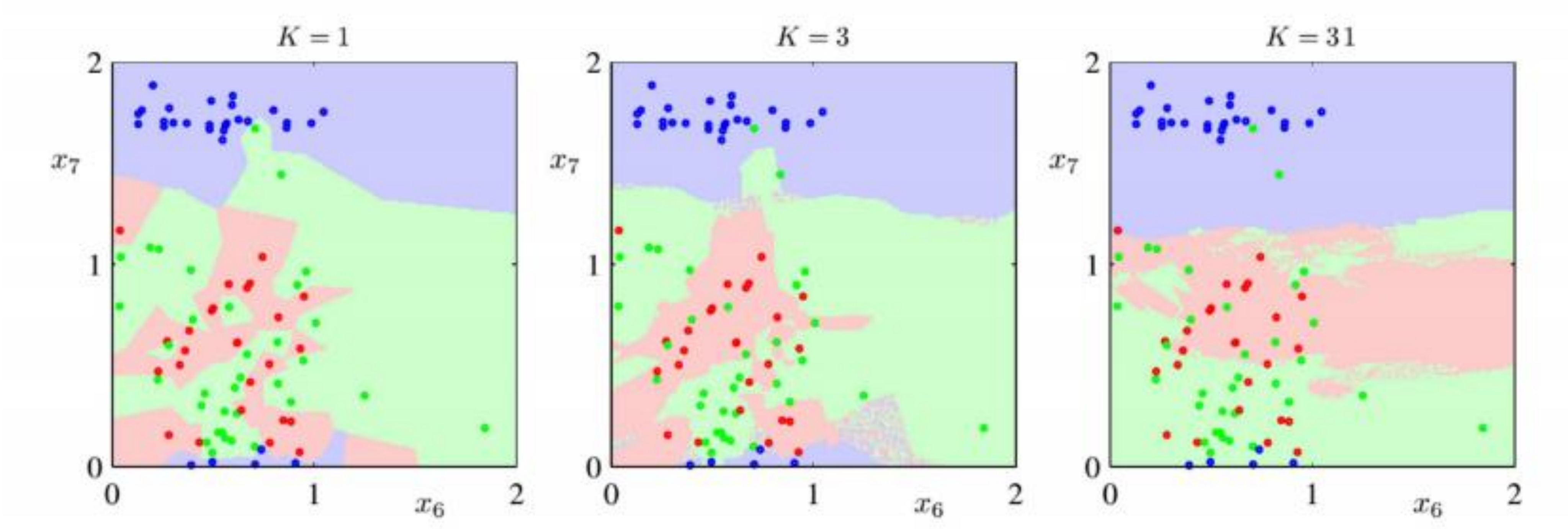
Example (Dynamic Query)

K-nearest neighbors for **new** point (k = 3)



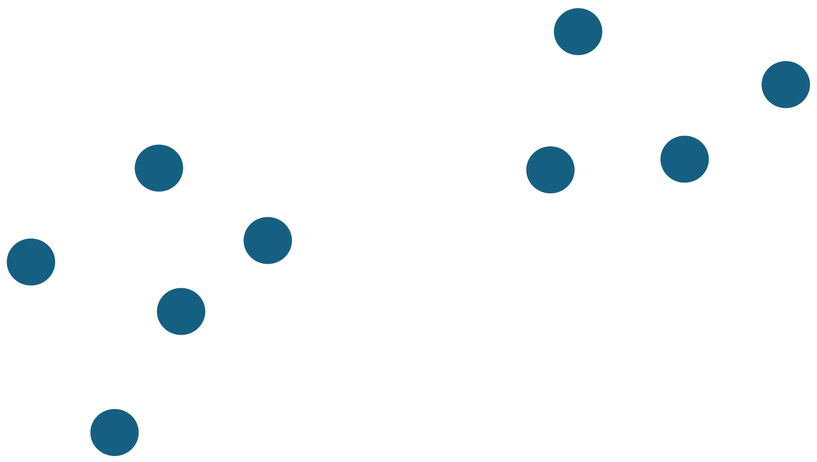
Example (Dynamic Query)

K-nearest neighbors for **new** point



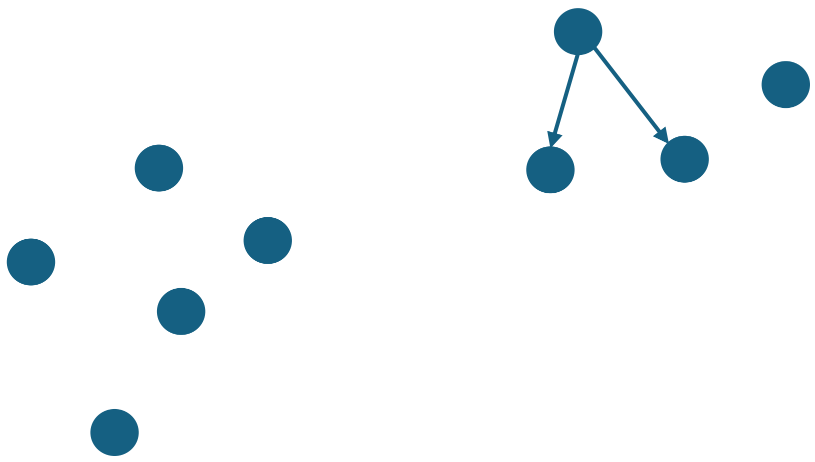
Example (K-Nearest Neighbor Graph)

K-nearest neighbors for every point (k=2)



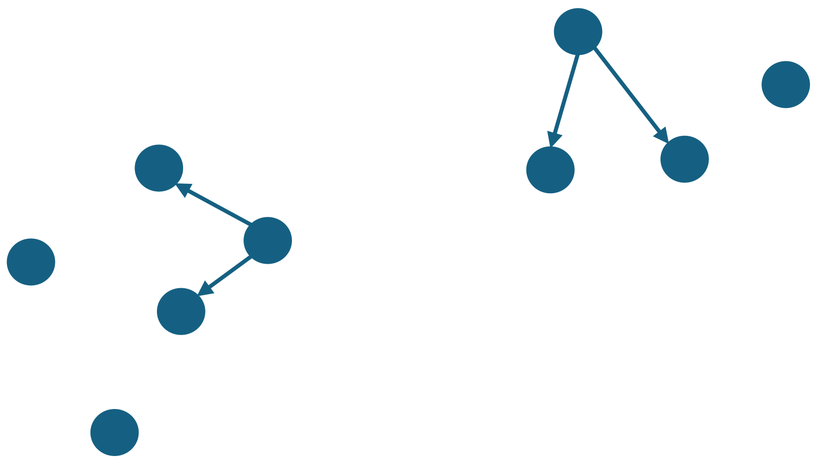
Example (K-Nearest Neighbor Graph)

K-nearest neighbors for every point (k=2)



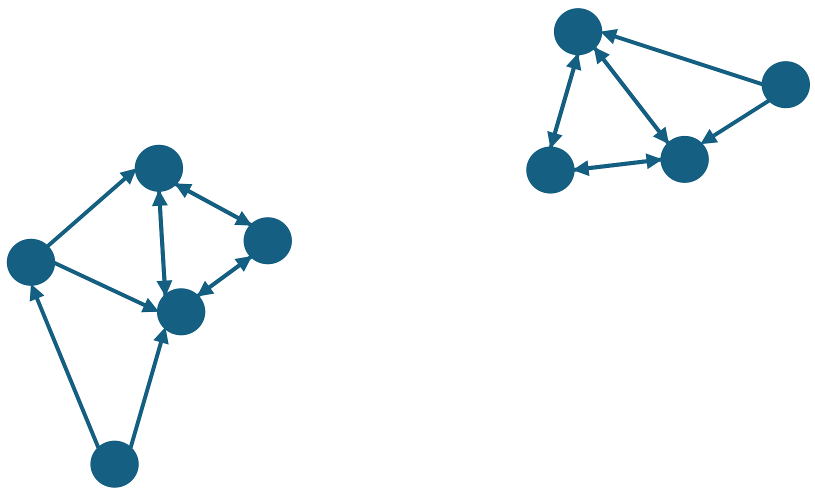
Example (K-Nearest Neighbor Graph)

K-nearest neighbors for every point (k=2)

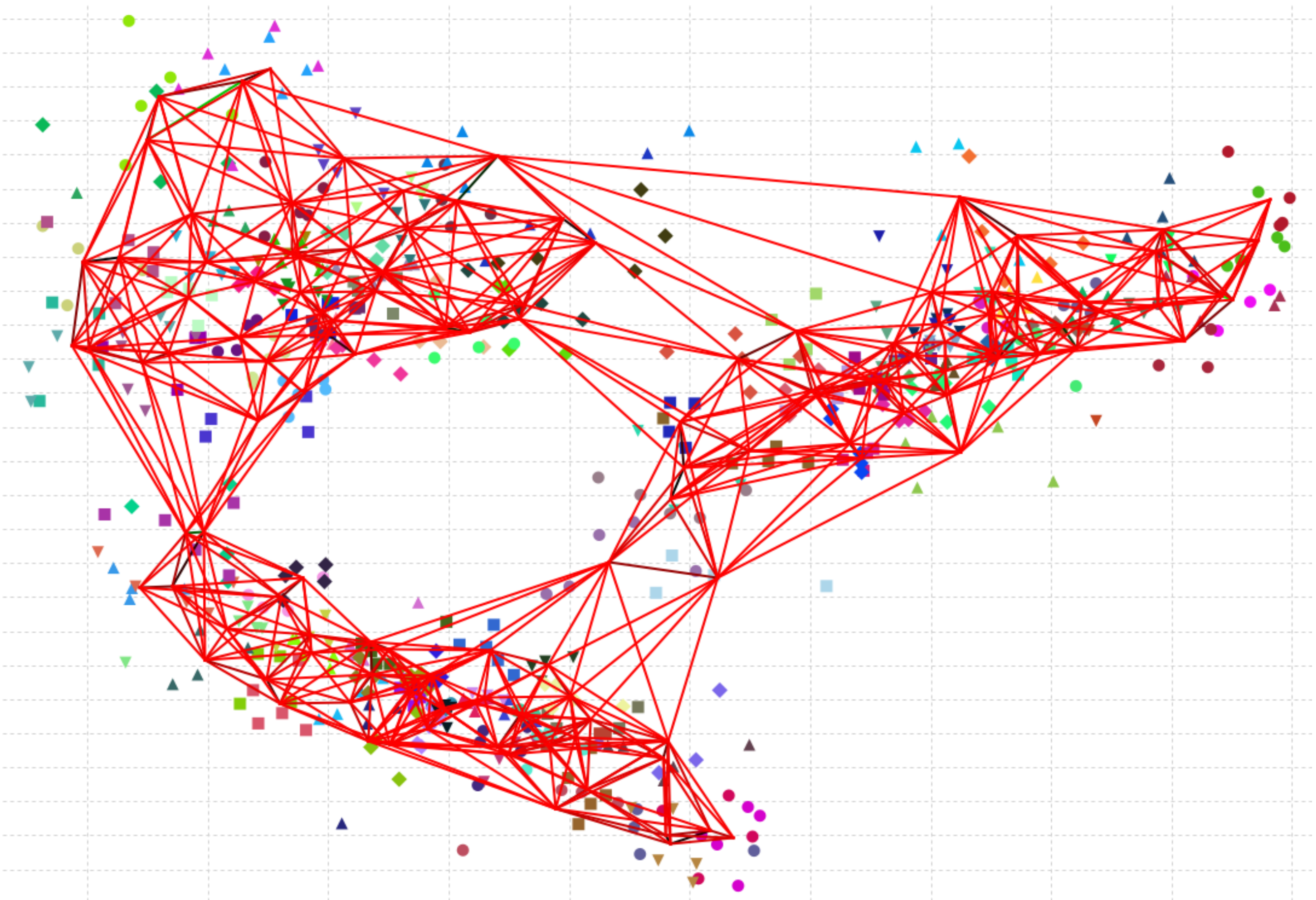


Example (K-Nearest Neighbor Graph)

K-nearest neighbors for every point (k=2)



Example (K-Nearest Neighbor Graph)

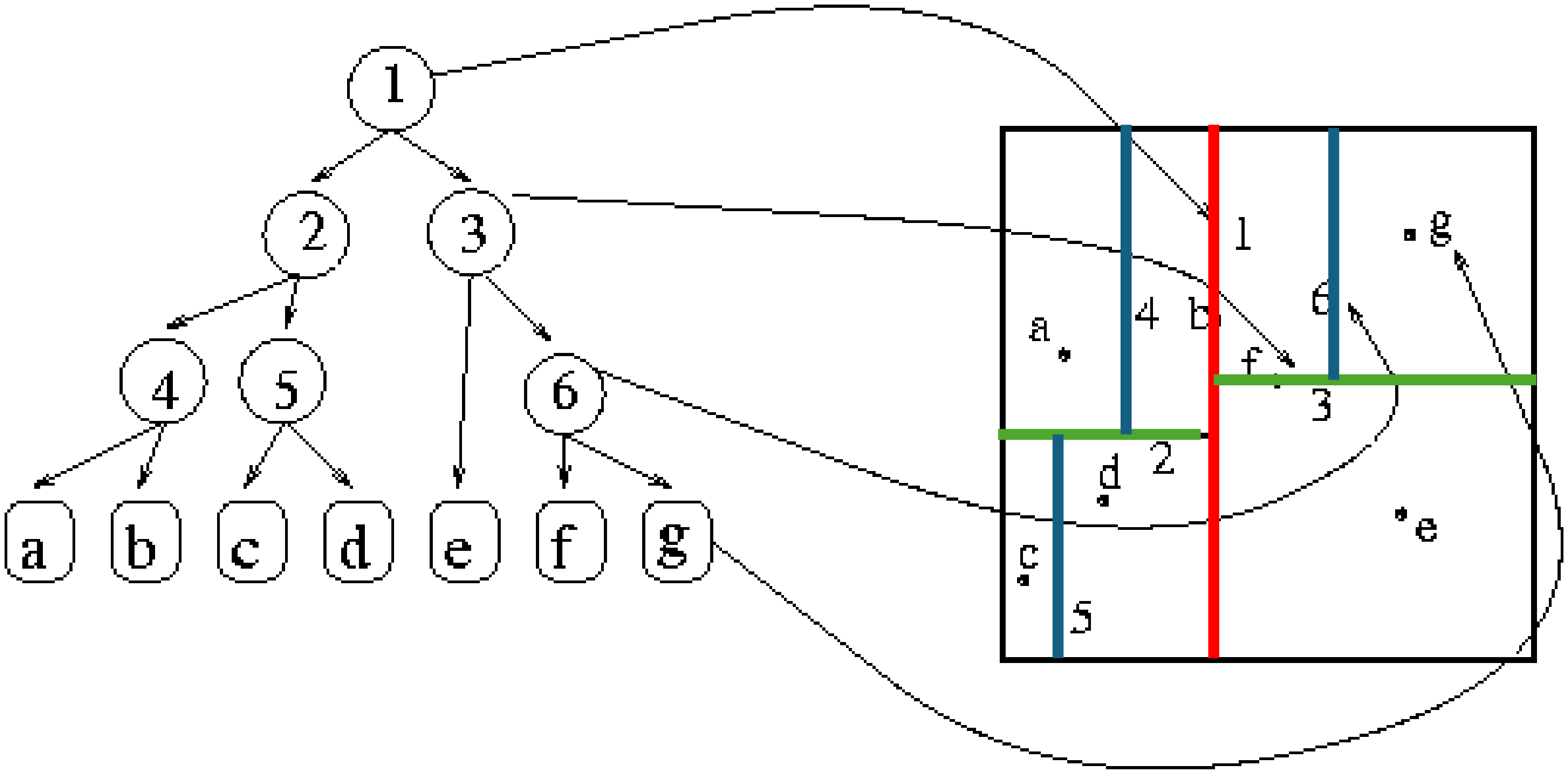


Relevant Preliminaries

Kd-Trees

- Tree data structure where each node represents a bounding box in d-dimensional space
  - Children split parent region into smaller bounding boxes at each level
- Common Splitting Rules:
  - Largest Dimension
  - Equal points on each side (median point)
- {quad,oct}-tree: 2^d equal sized children in d-dimensional space

Kd-Trees

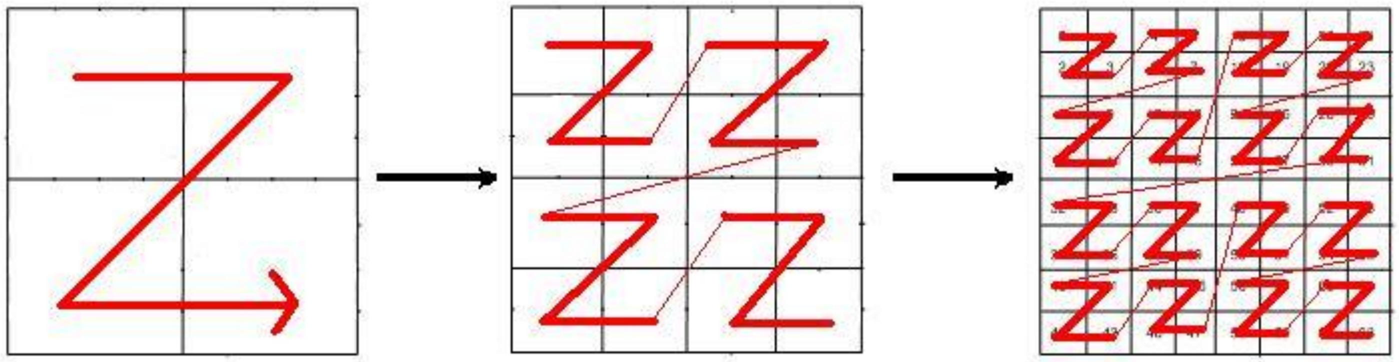


Morton Ordering

- Maps points from multidimensional space to one-dimensional sequence while preserving spatial locality
- Definition: Interleave the binary form of each integer coordinate

Morton Ordering

- Maps points from multidimensional space to one-dimensional sequence while preserving spatial locality
- Definition: Interleave the binary form of each integer coordinate



Existing Approaches to Nearest Neighbors

- Kd-Tree based algorithms
  - Construct kd-tree to break up d-dimensional space and explicitly explore
  - Slow construction, Fast Queries

- Morton based algorithms
  - Sort input by Morton order and traverse implicit tree
  - Fast construction, Slow Queries

Novel Idea

Zd-tree based algorithm

- Definition:
  - kd-tree whose splitting rule uses the Morton ordering
  - Root - Entire bounding box
  - Children - Split points at level i based on whether the bit at place i is 0 or 1
- Implementation:
  - Internal nodes stores corners of bounding box, 2 children, parent
  - Leaf nodes store constant number of points instead of children

Algorithm 1: buildTree(P, b)

Input: A set of randomly shifted points sorted according to their Morton ordering and an integer b representing the bit we are working on, starting with the highest bit.

Output: The leaf or internal node that contains P's bounding box

if $b == 0$ or size(P) $<$ sizeCutoff **then**
    return createLeaf(P)
else
    i = splitUsingBit(P, b) ;
    do in parallel
        L = buildTree($P[1 : i], b - 1$) ;
        R = buildTree($P[i : n], b - 1$) ;
    return createInternalNode(L, R) ;

Search Routines

- Goal: Add nearest neighbor candidates to min heap until certain all possible points have been explored
- Downward Search – Start at node and recurse downward into children
  - root-based algorithm starts at root with empty heap (single query)
- Upward Search – Start at leaf containing query point and recurse upward, performing downward search into sibling subtrees
  - bit-based algorithm finds leaf then calls searchUp (single query)
  - leaf-based algorithm runs searchUp from every leaf (full NN graph)

Downward Search

- Search if the furthest current candidate vertex is farther than the boxes bounding box from p
  - If leaf:
    - Check and add points to heap
  - Else:
    - Recurse on closer child bounding box
    - Recurse on further child bounding box

Algorithm 2: searchDown(T, p, N)

needed subroutines:

distance(p, N, k) returns infinity if N has fewer than k points and otherwise returns the distance from p to the furthest point in N; $k = 1$ if not specified.

insert(N, p, k) adds p to the set N keeping only the k closest points to p.

withinBox(T, p, r) returns true if p is within a distance r of the bounding box for T.

Input: A pointer to a tree node T, the query point p, and a current set of up to k nearest neighbors N.

Output: The k-nearest neighbors of p.

```
r ← distance(p, N, k) ;
if withinBox(T, p, r) then
    if T = Leaf then
        Q ← set of points contained in T ;
        for q ∈ Q do
            if q ≠ p then
                if distance(q, p) < distance(p, N, k)
                  then
                    insert(N, p, k);
    else
        R ← T.Right() ;
        L ← T.Left() ;
        ℓ ← distance(p, L.center());
        r ← distance(p, R.center()) ;
        if ℓ < r then
            N ’ = searchDown(L, p, N);
            return searchDown(R, p, N′);
        else
            N’ = searchDown(R, p, N);
            return searchDown(L, p, N′);
```

Upward Search

- Find leaf node p belongs to
- Add points in leaf to heap
- If furthest point is closer than edge of bounding box, return
- Otherwise, search down the sibling bounding box
- Recheck condition for parent bounding box

Algorithm 3: searchUp(C, p)
withinBox(T, p, r) with negative r returns true if p is in within the bounding box of T and at least r from the boundary.

Input: A leaf C of the kd-tree and a point p within the bounding box of C.

$N = \emptyset$
$Q \leftarrow$ set of points contained in C ;
for $q \in Q$ **do**
  if $q \neq p$ **then**
    if $d(q, p) <$ distance(p, N, k) **then**
      insert(N, p, k);
$r \leftarrow$ distance(p, N, k);
$P \leftarrow C$.Parent() ;
while not withinBox$(C, p, -r)$ and $P \neq \top$ **do**
  if P.Left() $= C$ **then**
    $N =$ searchDown$(P$.Right()$, p, N)$;
  else
    $N =$ searchDown$(P$.Left()$, p, N)$;
  $C = P$;
  $r \leftarrow$ distance(p, N, k);
  $P \leftarrow C$.Parent() ;
return N

Batch Dynamic Updates

- Recursively insert into children in parallel
- Split leaves if insertion would exceed leaf size restriction

Algorithm 4: batchInsert(T, P)

Input: A pointer to a node T of the kd-tree and a set of points P contained in its bounding box and sorted according to their Morton order

if T = Leaf **then**
    if size(P) + size(T) ¡ leafCutoff **then**
        Insert $p \in P$ into T ;
    else
        Split T into multiple leaf nodes ;
else
    $b = T \rightarrow$bit ;
    i = splitUsingBit(P, b) ;
    do in parallel
        batchInsert($T \rightarrow$ Right, $P[1 : i]$) ;
        batchInsert($T \rightarrow$ Left, $P[i : n]$) ;

Theorem 2.1

For a point set P of size n with bounded ratio, the zd-tree can be built using $O(n)$ work with $O(n^{\epsilon})$ span and resulting tree height $O(\log \mathrm{n})$

Proof

- Bounding cube of P has side length within constant factor of max distance between two points
- Must be divided until two points minimum distance from one another are in separate cubes (assuming no coarsening)
- dmax / dmin = poly(n) due to bounded ratio assumption
- Tree has $O(\log \mathrm{n})$ depth
- This implies radix sort can be used since we only need compare $O(\log \mathrm{n})$ bits to construct the tree

Theorem 2.2

For a zd-tree representing a point set P of size n with bounded expansion, finding the k-nearest neighbors of a point $p \in P$ requires expected $O(k \log \mathrm{k})$ work.

Proof

- Work separated into two parts:
  - Work of searching through the points in leaves of the zd-tree
  - Work of finding candidate leaves

Lemma 2.1

- Searching for the k-nearest neighbor of a point p
  - O(k) candidate points will be considered
  - Resulting in O(klogk) work to evaluate candidate points

Lemma 2.1

- Searching for the k-nearest neighbor of a point p
  - O(k) candidate points will be considered
  - Resulting in O(klogk) work to evaluate candidate points

Proof

- Find ancestor box B with O(k) points
- Expand radius to 4r
- All candidate points contained in box
- Constant number of “overlapping” leaves

Lemma 2.2

- Traversing zd-tree
  - Expected number of tree edges to find nearest neighbors is O(k)

Lemma 2.2

- Traversing zd-tree
  - Expected number of tree edges to find nearest neighbors is O(k)

Proof

- Lemma 2.1 established search space contains O(k) points
- Each edge we traverse to visit a leaf can be amortized against the total number of leaves

- Length of longest path is O(1) in expectation

Theorem 2.3

Let T be a pruned zd-tree representing point set P, and let Q be a point set of size k, such that |P| + |Q| = n. Then if P ∪ Q and Q both have bounded expansion and bounded ratio in the same hypercube X, Q can be inserted into T in $O(k \log(n/k))$ work and $O(k^{c} + polylog(n))$ span.

Implementation

- C++ using ParlayLib
- Optimizations:
  - Squared distances
  - Vector for small k and C++ stl priority queue for large k
  - Sort queries according to Morton order to improve cache utilization

Empirical Results

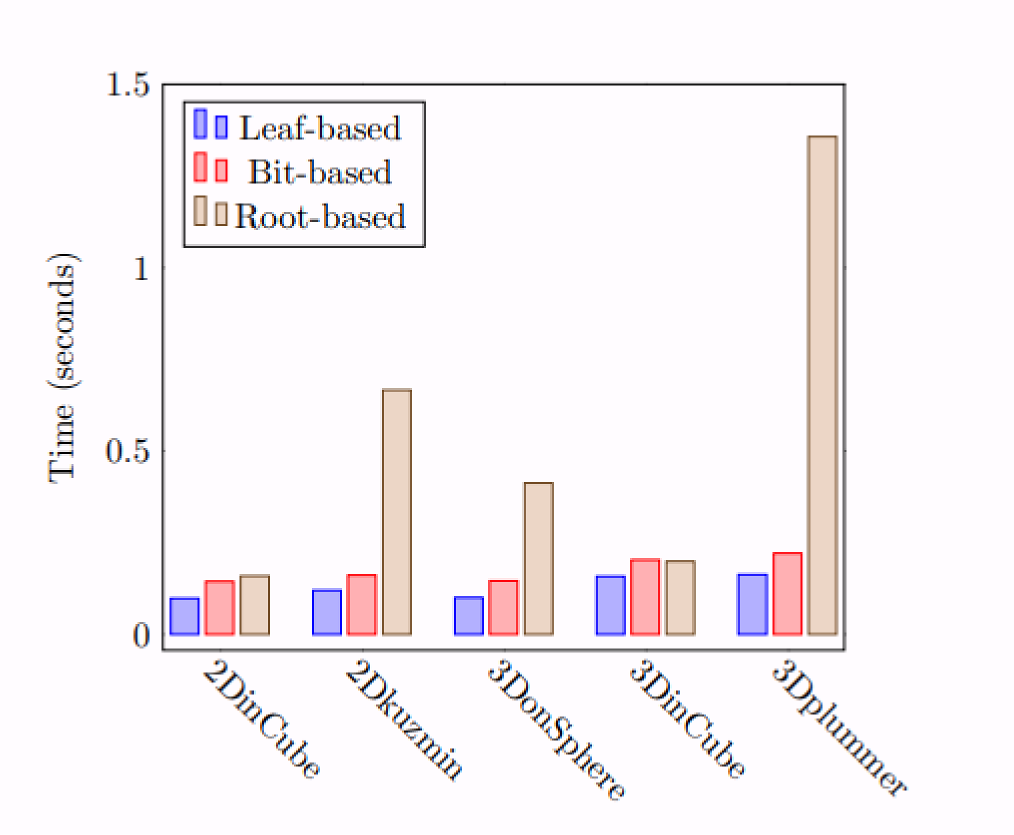
Benchmarked against 3 existing implementations

- Chan: Naïve search from root
- STANN: k-nearest neighbors and graph functions
- CGAL: Widely adopted parallel implementation

- Note: Authors updated Chan and STANN to utilize modern parallel primitives

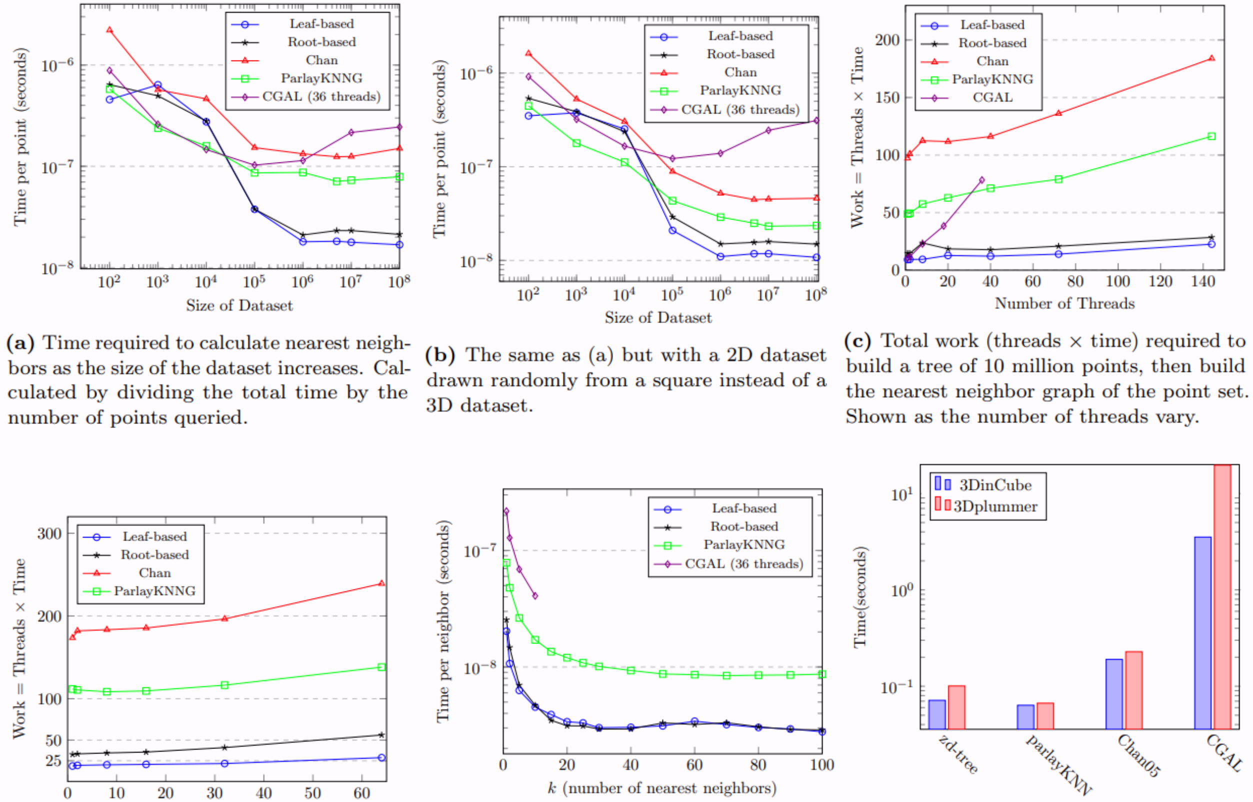
Search Strategies

- Compared three search strategies on four datasets of 10 million points
- Root based algorithms perform poorly
- Leaf based algorithm barely beats bit-based despite O(n) vs O(nlogn) work because constant factor is much larger



Comparison to State-of-the-Art

- Both implementations faster (often by order of magnitude)
- Robust performance across data sets and k
- Work efficiency
- Good scalability
  - 75x speedup with 72 cores and 144 hyperthreads



(a) Time required to calculate nearest neighbors as the size of the dataset increases. Calculated by dividing the total time by the number of points queried.

(b) The same as (a) but with a 2D dataset drawn randomly from a square instead of a 3D dataset.

(c) Total work (threads × time) required to build a tree of 10 million points, then build the nearest neighbor graph of the point set. Shown as the number of threads vary.

(d) The same measurements as (c), but on the 32-core AMD machine.

(e) Time required to calculate a neighbor as the number of neighbors k increases. Calculated by dividing the total time by k times the number of queries.

(f) The bars represent the total time each algorithm takes to build the data structure, for points drawn randomly from a 3D cube, and points drawn from a Plummer distribution.

Figure 4: Statistics related to non-dynamic queries. Unless otherwise stated, the size of the dataset is 10 million, the number of nearest neighbors $k = 1$, experiments were performed on 144 threads on a 72-core Dell R930, and data points are drawn randomly from a 3D cube.

Strengths

- Solid performance and scalability
- Elegant idea bridging existing strategies
- Support for parallel batch dynamic insertions and deletions

Weaknesses

- Requires bounding box of all data to be specified before hand
- Proofs are difficult to follow
- Tradeoff between work and span for sorting strategies

Future Directions

- Extensions to higher dimensions
- Relaxing theoretical assumptions
- Concurrent updates and queries
- External memory

Discussion Questions

- How realistic are theoretical assumptions for real world data?
- Would this implementation scale well on GPUs?
- What is the cost of preprocessing? How does it compare to existing implementations?