Multilateration Indexing

Data structures and algorithms to improve query performance for geodesic and other complex distance functions

Chip Lynch, University of Louisville, Summer, 2021

Agenda

1. Hi, I’m Chip!
2. Problem Statement
3. Multilateration Index
4. Query Algorithms
5. Experiments
6. Results
7. Conclusions and Future Work

Hi, I’m Chip!

- Came back to grad school for “fun”
  - It’s been fun!
  - Did some research, took some classes. Good times.
- Data and Software Engineer for:
  - KPMG / Deloitte
  - NASA
  - US Air Force
  - SpaceX
  - Passport Health (Medicare/Medicaid)

Problem Statement

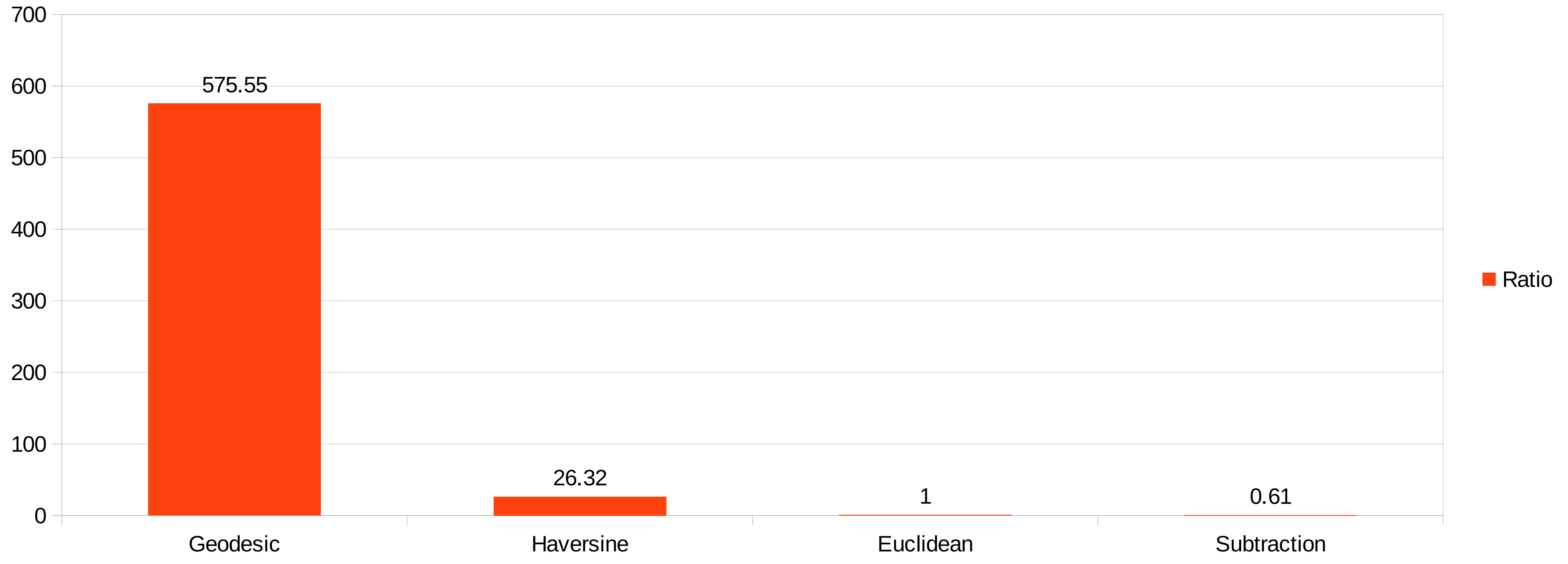
Real World Problems

- Satellite Communications
  - Fast Moving
  - Extreme accuracy requirement over long distances for laser guidance
  - Regularly need to find the closes Satellite/Ground Station/User Terminal (Nearest Neighbor)

- Healthcare Network Adequacy
  - Ensure that **x%** of **<type of person>** is within **m** miles of **<type of health service>**
    - 80% of female members over the age of 13 must live within 25 miles of an OB/GYN
    - 80% of members under the age of 16 must live within 25 miles of a pediatrician
    - 90% of members must live within 50 miles of an emergency room
    - ...

Problem Statement

Expensive Distance Functions



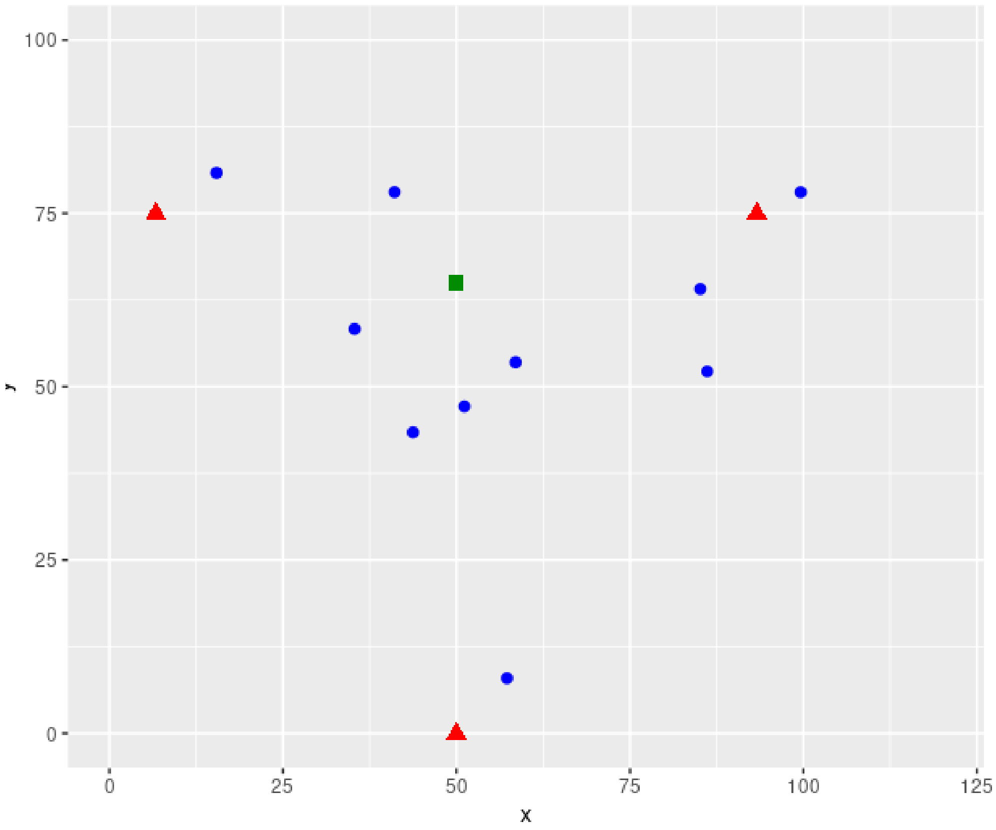
Problem Statement

Network Adequacy

- Given a non-empty set of points P and a non-empty set of query points Q in a metric space M (where P ∩ Q comprises the 'network'), the network is '**completely adequate**' for a distance d and a distance function D(a, b) describing the distance between points a and b for a ∈ M and b ∈ M if for every point q (where q ∈ Q) ∃ at least one point p (p ∈ P) 3 D(p, q) <= d. Otherwise the network is '**inadequate**.'
- We call a single point q '**adequate**' itself, if it satisfies the same condition – i.e. ∃ at least one point p (p ∈ P) D(p, q) <= d.
- If, within P , we consider the largest subset P' ∈ P where P' is 'completely adequate,' then P has a "**Network Adequacy Percent (NAP)**" of |P'|/|P|. Note that P' can be defined (identically) as the union of all 'adequate' points p ∈ P

Problem Statement

Nearest Neighbor



Given a set of search points P (blue dots), and a query point Q (green square), determine which of the points in P are closest to the query point Q

(Advanced) Nearest Neighbor

Problem Statement

- Constraints beyond typical NN problems:
  - Constant motion – pre-processing must happen repeatedly and quickly
  - Expensive distance function D(a, b)
  - Queries specify subsets of points in P and Q
- Includes “kNN”:
  - Find the nearest k neighbors out of the set P

Problem Statement

Comparing Solutions

1. Training Time Complexity
   - the O() (typically in terms of |P|) required to pre-process the points p if any
2. Memory Space
   - the memory requirements of the structures resulting from pre-processing
3. Prediction Time Complexity
   - the O() required to find the closest point $p \in P$ for a single query point q
4. Insertion/Move Complexity
   - the O() complexity required to add or move (or remove) a point $p \in P$ (or $q \in Q$)

Comparing Solutions

Problem Statement

1. Training Time Complexity
   – the O() (typically in terms of |P|) required to pre-process the points p if any
2. Memory Space
   – the memory requirements of the structures resulting from pre-processing
3. Prediction Time Complexity
   – the O() required to find the kN N ∈ P for a single poin

Most solutions optimize for #3

4. Insertion/Move Complexity
   – the O() complexity required to add or move (or remove) a point p ∈ P

Existing NN Solutions

Problem Statement

1. Brute Force
   - Compare every point to each other and sort the resulting distances
2. Space Partitioning Trees
   - Recursively divide points into smaller and smaller areas to reduce search space
   - Examples: k-d trees and ball trees
3. Locality Sensitive Hashing
   - Project points to lower dimensions while attempting to maintain distance relationships
4. Graph Based Search
   - Create a network of locally nearest-neighbors
   - Example: Facebook’s FAISS

Proposing a Solution

- The Multilateration Index
  - Store distances from fixed points, rather than coordinates
  - i.e. Store distances from three points around the globe for geospatial applications
- Benefits:
  - Allows direct comparison of distances using simple subtraction
  - Can be quickly calculated and updated (particularly vs. graph and space partitioning)
  - Allows error checking (many combinations of distances are impossible, unlike coordinates)
  - Is easily augmented with additional data that does not affect optimization

Example Index:

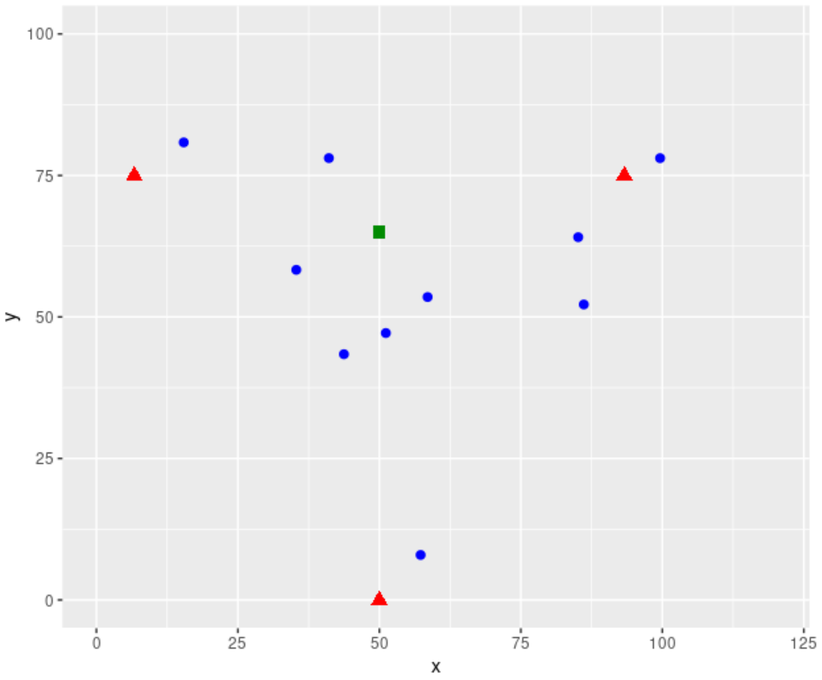
Index	x	y	d1	d2	d3
8	15.42852	80.83634	10.50105	78.09115	87.91872
4	35.32139	58.321179	33.12763	60.331164	60.14001
6	41.08036	78.065907	34.51806	52.310835	78.57382
2	43.7394	43.418684	48.67639	58.768687	43.86772
7	51.14533	47.157734	52.44704	50.520446	47.17164
1	58.52396	53.516719	56.10157	40.877779	54.1913
9	85.13531	64.090063	79.19169	13.627535	73.08916
5	86.12714	52.201894	82.6355	23.900247	63.48392
3	57.28944	7.950161	83.99465	76.108693	10.78615
10	99.60833	78.055071	92.95982	7.008032	92.48557

Example Walkthrough Step 1:

3 Red Triangles:
Reference Points

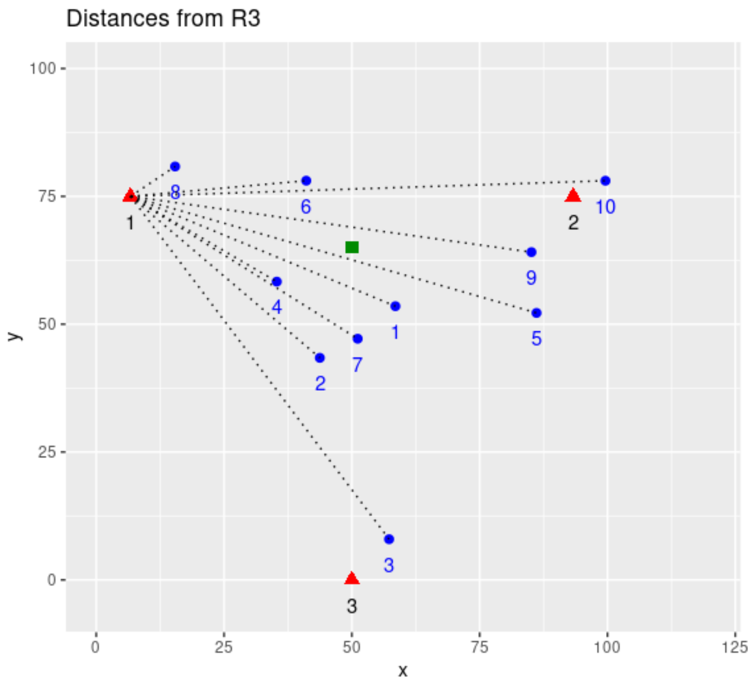
10 Blue Dots:
Search Points

1 Green Square:
Query Point

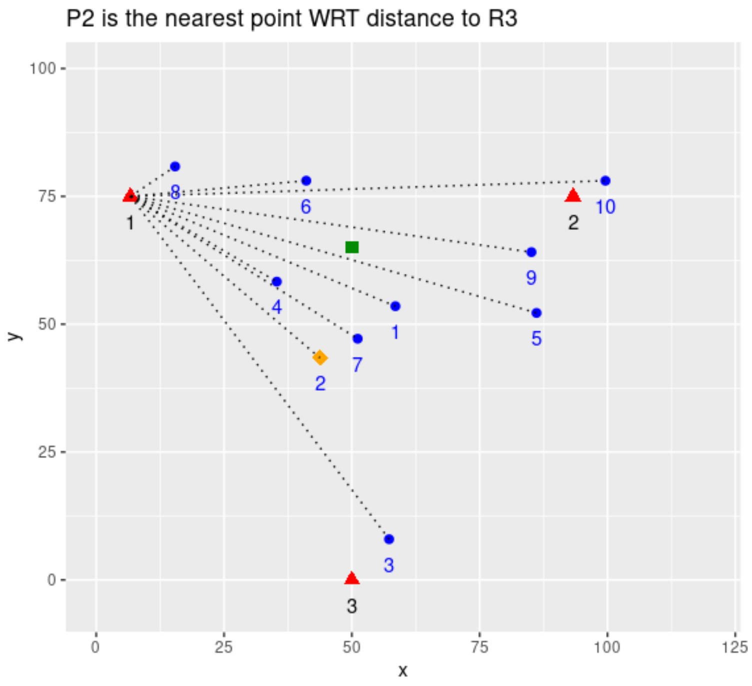


Example Walkthrough Step 2:

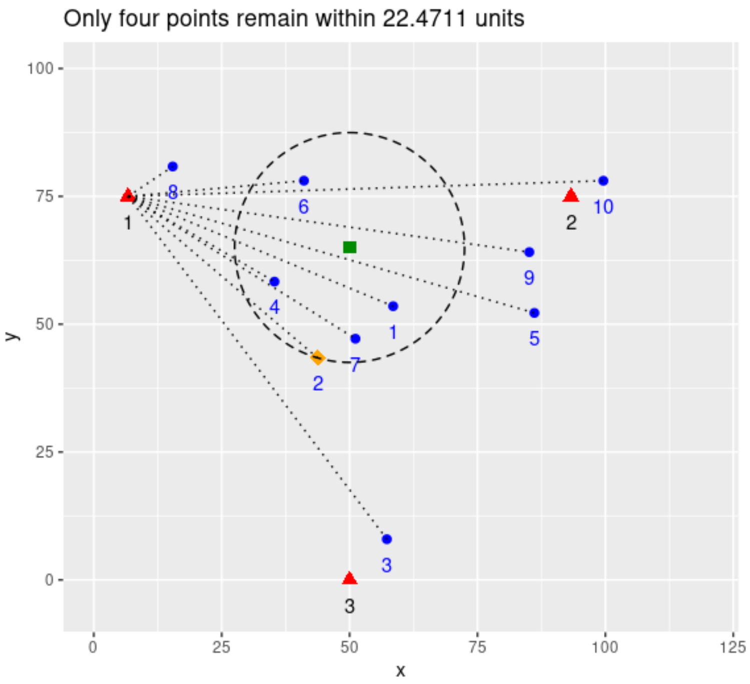
Algorithms



Example Walkthrough Step 3:

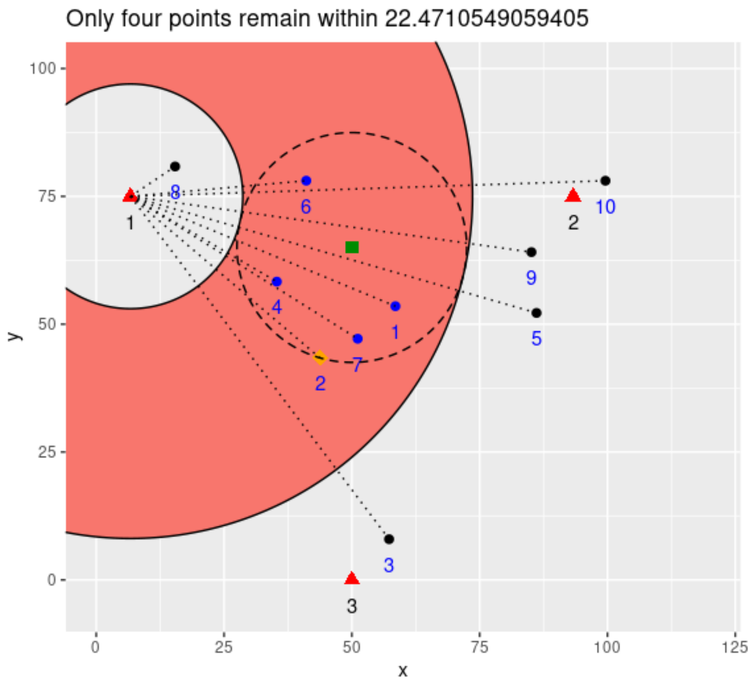


Example Walkthrough Step 4:

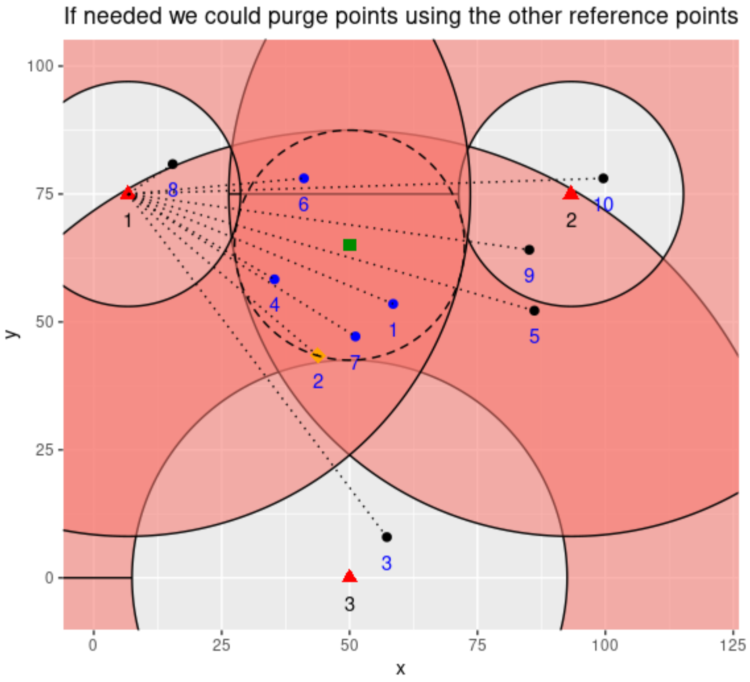


Algorithms

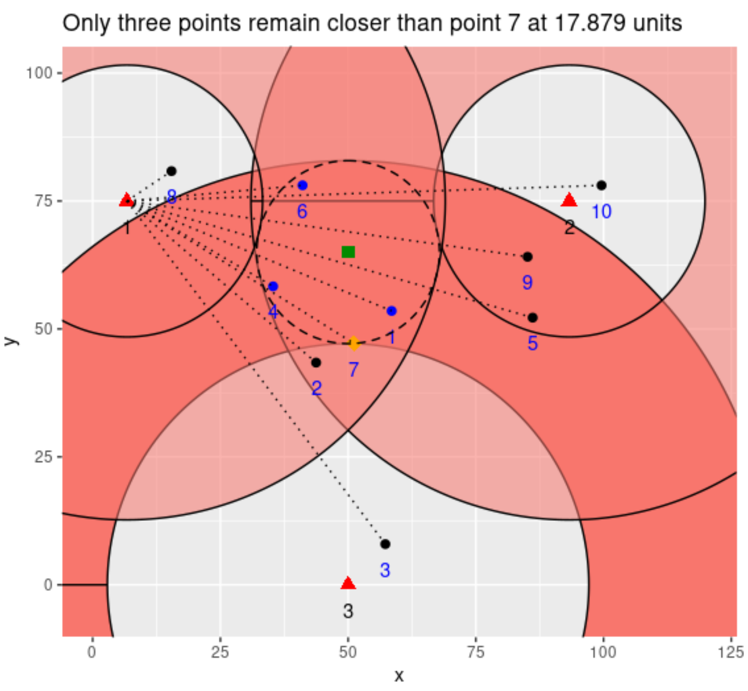
Example Walkthrough Step 5:



Example Walkthrough Step 6:

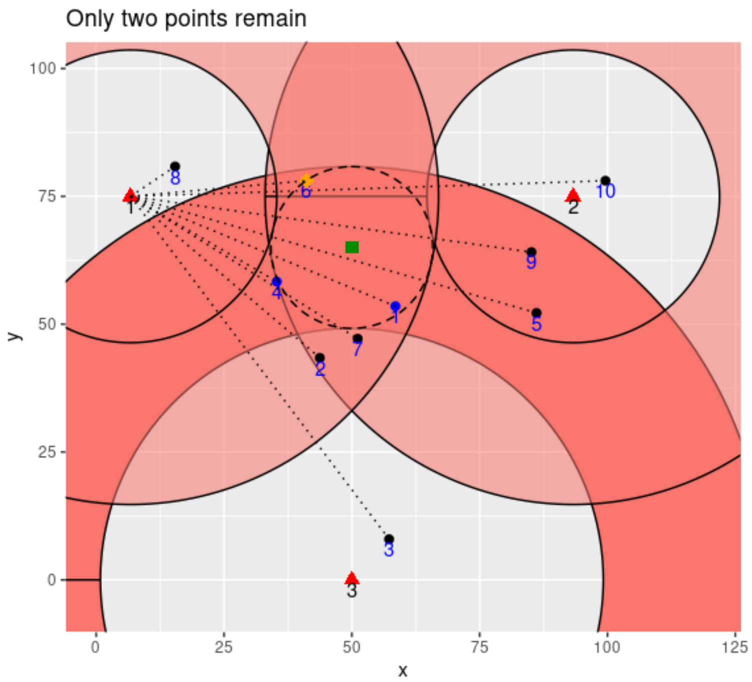


Example Walkthrough Step 7:



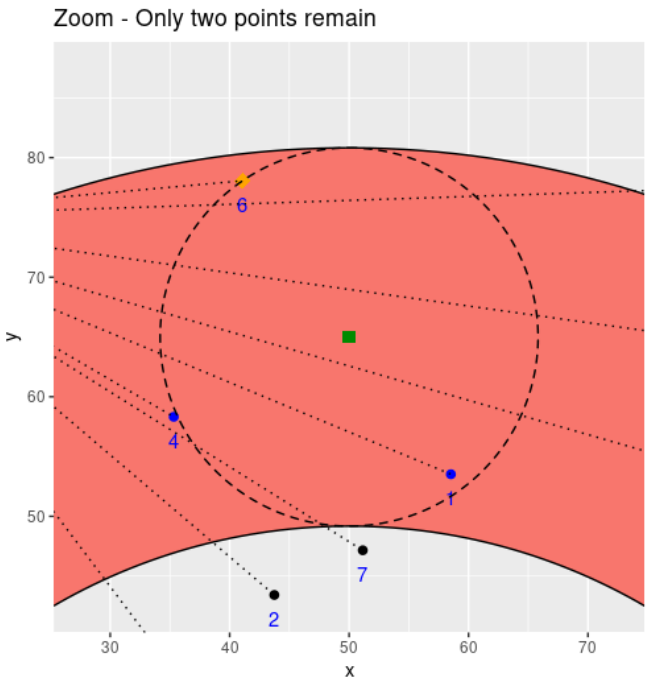
Example Walkthrough Step 8:

Algorithms



Algorithms

Example Walkthrough Step 9:



Experiment Setup

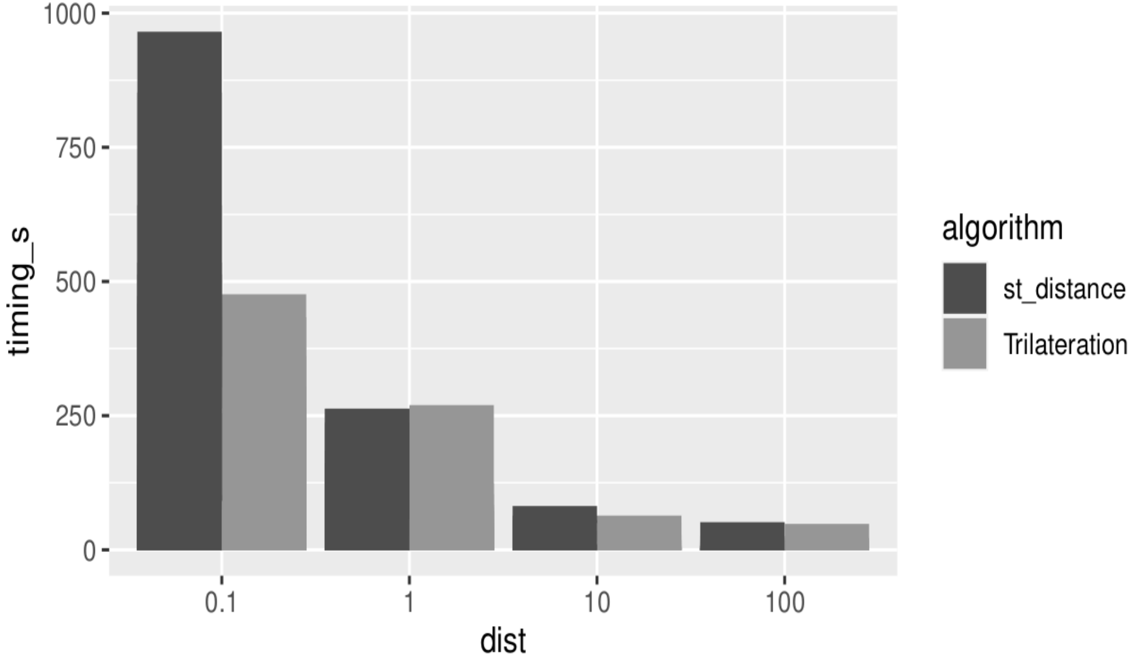
Nearest Neighbor:

- Use Open Source “ANN” Software
- Modify popular Scikit-Learn to run our algorithms
- Use custom geodesic data set (since none already existed)
- Use existing Euclidean and Angular data sets for comparison

Network Adequacy:

- Use Postgresql due to geodesic distance support
- No strong alternatives exist in SQL so we compare to naive use of “st_distance”

SQL NA Timings:



Good:

- 50% faster on sparse data (NAP <~ 50%)

Bad:

- No major difference for dense data (NAP >~ 80%)

Results

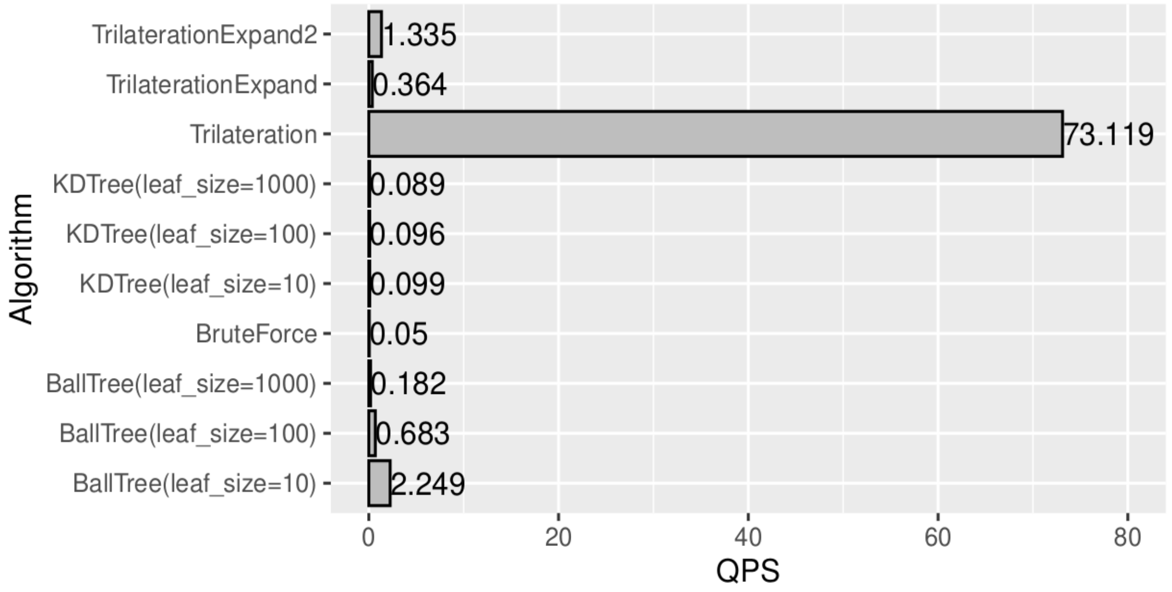
Cython Nearest Neighbor

Good:

- 50x faster on Geodesic Nearest-Neighbor queries than other algorithms
- Also fast pre-processing times not included in the 50x value

Bad:

- Horrible performance on Euclidean and other simple distance functions



Objectives Met!

- We were able to improve performance in SQL and Scikit-Learn for complex distance functions
- We are reasonably optimized for training time, memory space, and insert/move time complexity as well as query performance
- We did NOT create a magical algorithm that improves in the universal case, but we have good bounds on where our algorithm will be useful in advance

So many things...

There are many, many avenues of research we think this opens up for future work:

- This is new code and can likely be optimized:
  - Parallelization should be easy but distributed computing versions were not implemented here for compatibility with our testing libraries
  - Precision is extremely high; in fact with 64-bit distances and meter units, we are precise to atomic levels measuring on earth. This is unnecessary and probably slows us down.
- The Index values themselves have other uses…
  - These can be used as an alternate coordinate system instead of Lat/Long
  - “Geohashing” based on Trilateration distances could be valuable
- Search for other domains with complex distance functions:
  - Protein folding? Recommendation Systems? Cryptography?
  - Non-metric space applications?