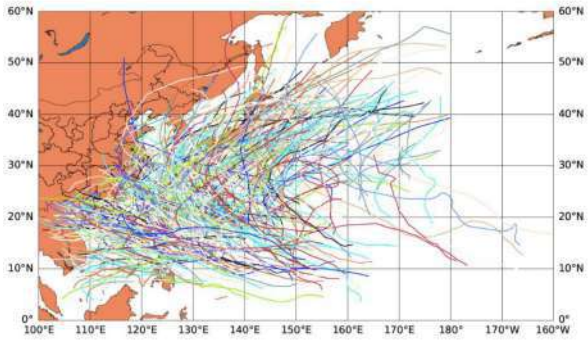
NC STATE UNIVERSITY

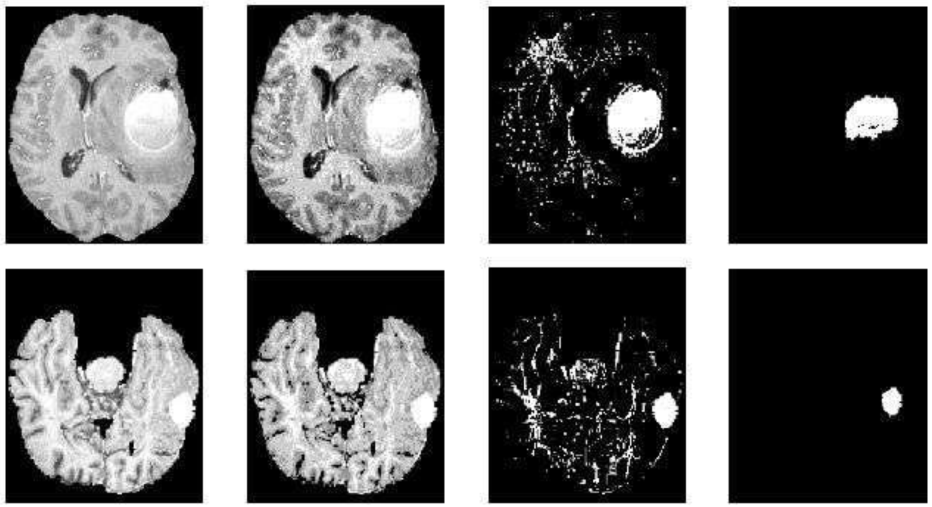
Computing the Hausdorff Distance

Parth Parikh

Masters Thesis Defense, 05/01/23

Applications of Hausdorff Distance





- Find similar cyclone trajectories from set of historic cyclone trajectories
- Analyze migration trajectories of different types of birds

Nutanong et al.

- Evaluation of 3D brain tumor segmentation (identifying tumor regions from MRI)

Khotanlou et al.

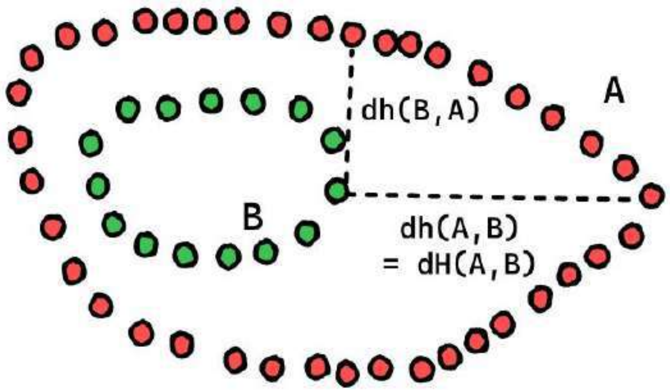
What is Hausdorff Distance?

- The **directed Hausdorff distance** between sets `A` and `B`:

$$\mathbf{d}_h(A, B) := \max_{a \in A} \mathbf{d}(a, B).$$

- The **undirected Hausdorff distance** is the larger of the directed Hausdorff distances:

$$\mathbf{d}_H(A, B) := \max\{\mathbf{d}_h(A, B), \mathbf{d}_h(B, A)\}.$$



Introduction and Contributions

- Survey of Hausdorff Distance
- Greedy Trees
- Hausdorff Distance using Greedy Trees

Papers

- Proximity Search in the Greedy Tree
  - *Symposium on Simplicity in Algorithms (SOSA), 2023*
- Linear-Time Approximate Hausdorff Distance
  - *The 30th Fall Workshop on Computational Geometry, 2022*
- Approximating the Directed Hausdorff Distance
  - Submitted to *35th Canadian Conference on Computational Geometry, 2023*
- Greedy Permutations and Finite Voronoi Diagrams
  - Submitted to Multimedia Exposition in Computational Geometry, SoCG 2023
- The Finite Voronoi Method
  - Prepared for resubmission

Survey of HD Techniques for Point Sets

Naive Algorithm for Directed Hausdorff Distance

```
procedure DIRECTED_HAUSDORFF_DISTANCE(A, B):

LB = 0
for a in A:
    UB = float("inf")

    for b in B:
        UB = min(UB, d(a, b))

    LB = max(LB, UB)

    return LB
```

Survey of HD Techniques for Point Sets

- Previous surveys?
- Improvements on the naive Quadratic Algorithm
  - The Inner Loop is Nearest Neighbor Search
  - Nearest Neighbor Search is Overkill (The Early Break Heuristic)
  - Ordering Matters
  - There is Spatial Locality in the Searches
  - Preprocessing Allows for Efficient Branch and Bound Algorithms
  - Outer Loop Pruning
- <u>Most</u> heuristics were used for *constant factor* improvements

The Inner Loop is Nearest Neighbor Search

```
procedure DIRECTED_HAUSDORFF_DISTANCE(A, B):

LB = 0
for a in A:
    UB = float("inf")

    for b in B:
        UB = min(UB, d(a, b))

    LB = max(LB, UB)

    return LB
```

```
OBSERVATION:
inner loop
sequentially computes
the nearest neighbor distance
of each point in A
```

Nearest Neighbor Search is Overkill (The Early Break Heuristic)

```
procedure DIRECTED_HAUSDORFF_DISTANCE(A, B):

LB = 0
for a in A:
    UB = float("inf")

    for b in B:
        UB = min(UB, d(a, b))

    LB = max(LB, UB)

    return LB
```

OBSERVATION:
If
d(a, b) < LB
in the inner loop,
we can break early!

Ordering Matters

```
procedure DIRECTED_HAUSDORFF_DISTANCE(A, B):

LB = 0
for a in A:
    UB = float("inf")

    for b in B:
        UB = min(UB, d(a, b))

    LB = max(LB, UB)

    return LB
```

OBSERVATION:
Traversal order important for reducing computations with spatial correlation

Use randomization or greedy permutation

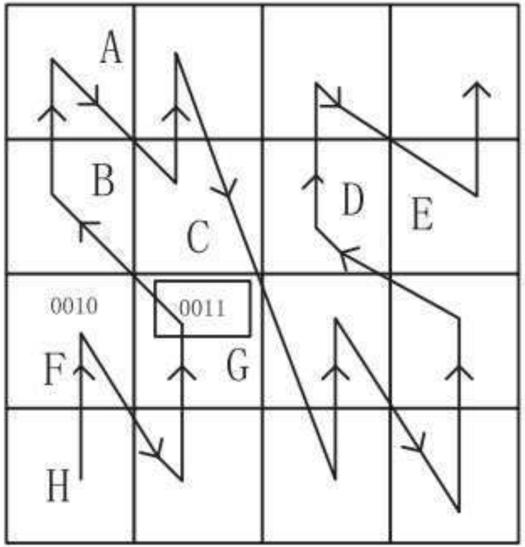
Foreshadowing HD using Greedy Trees

There is Spatial Locality in the Searches

```
procedure DIRECTED_HAUSDORFF_DISTANCE(A, B):

LB = 0
for a in A:
    UB = float("inf")

    for b in B:
        # Early Break
        if d(a, b) < UB:
            break

        UB = min(UB, d(a, b))

    LB = max(LB, UB)

    return LB
```

OBSERVATION:
The location of previous early break can be stored.

Execution can continue from where we previously left before



Morton Curves for mapping multidimensional point sets to 1D

Preserves locality of data points

Preprocessing Allows for Efficient Branch and Bound Algorithm

```
IDEA #1:
1. Partition A into
(A1, A2, ... , At)

dh(A, B) = max(dh(Ai, B))
for i from 1 to t

2. For each part in partition
compute the UB and LB to B

Ai ⊆ Si = ball(c, r)

d(c, B) + r ≥ dh(Ai, B) ≥ d(c, B)

3. If the UB < LB,
we can prune the part!
```

```
procedure DIRECTED_HAUSDORFF_DISTANCE(A, B):

LB = 0
for a in A:
    UB = float("inf")

    for b in B:
        UB = min(UB, d(a, b))

    LB = max(LB, UB)

return LB
```

```
IDEA #2:
1. Partition B into
(B1, B2, ... , Bt)

2. For each part in partition
compute the UB and LB to a

Bi ⊆ Si = ball(c, r)

d(a, c) ≤ dh(a, Bi) ≥ d(a, c) + r

3. Pruning
```

```
Observation: Use both Idea #1 and #2
```

Outer Loop Pruning

Sample B' from B

run DIRECTED_HAUSDORFF_DISTANCE(A, B')
st. for each b' in the inner loop
keep track of its NN seen so far

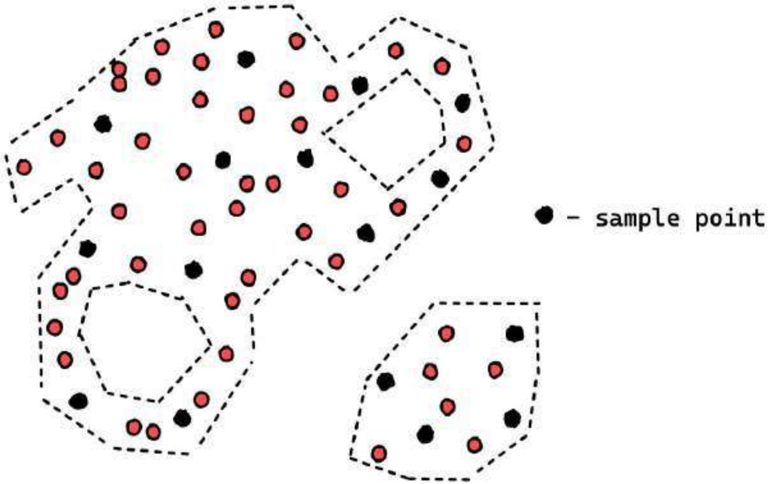
using LB, and NN distances

run DIRECTED_HAUSDORFF_DISTANCE(A, B)
st.
if at an iteration, b in B'
AND
NN(b, A) < LB, skip the iteration!

Greedy Permutation
and
Greedy Trees

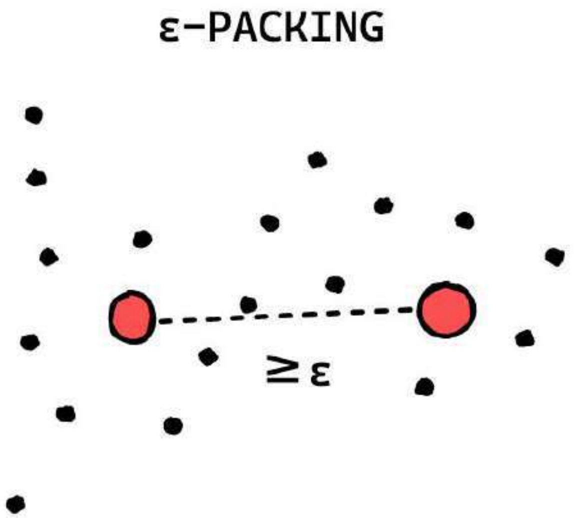
What is a Good Sample?

- A good sample can be observed using Hausdorff distance.

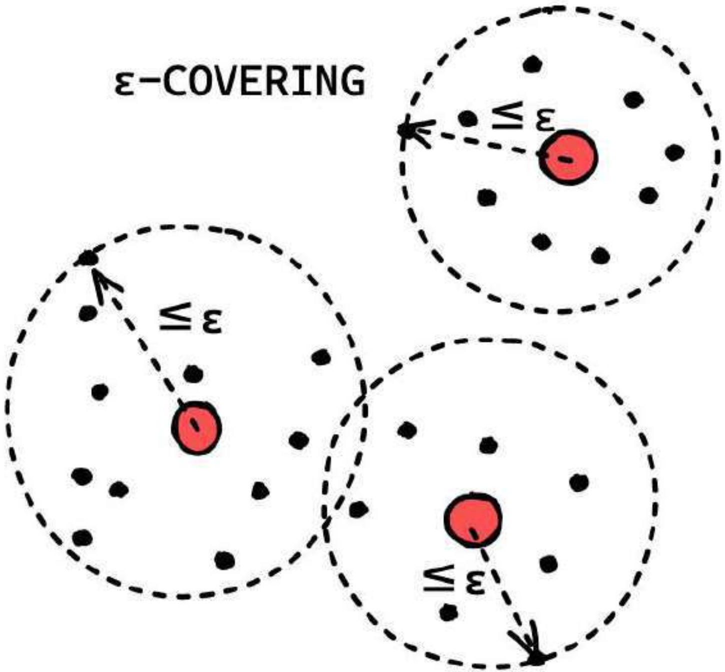


- We would like to capture the **geometry of the original set** with far fewer points
  - How?
  - Using two properties: **Packing** and **Covering**

Packing, Covering, and ε-Net



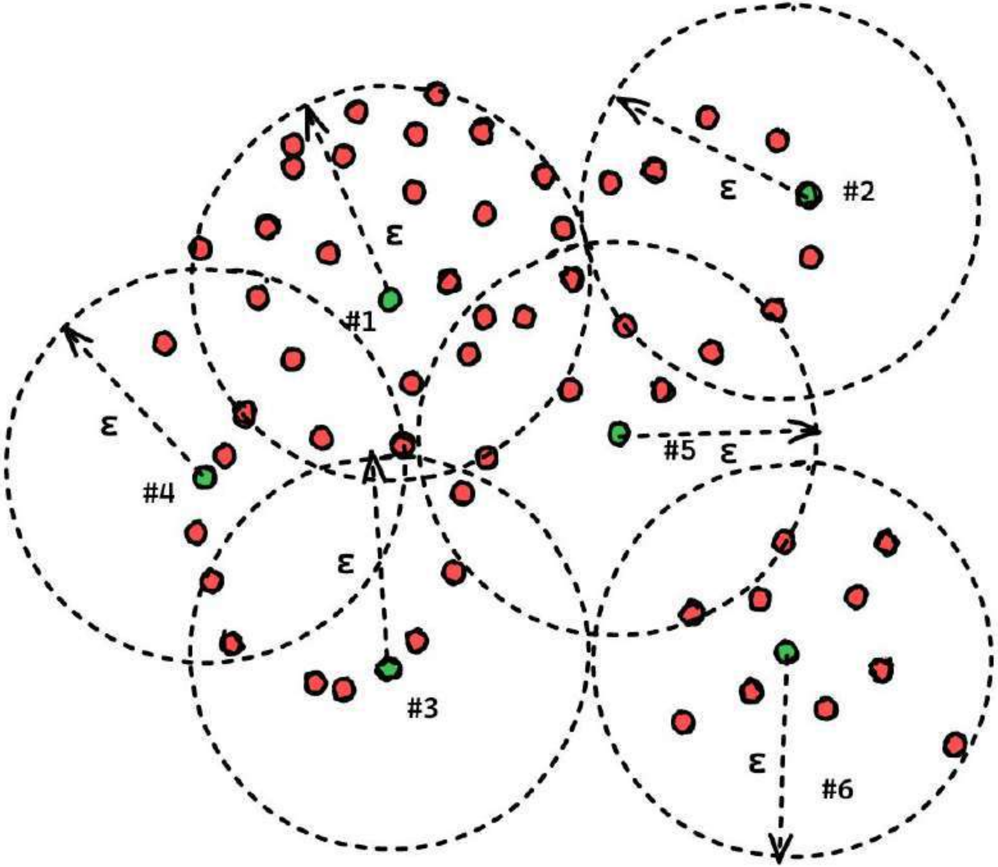
All pairs of sample points
are ε-apart



ε-radius balls centered at sample points
that cover the set

Having seen that, we ask again, <u>what is a Good Sample</u>?

- Has an ε-net for some good ε
- If we know what ε is, we can construct it as:
- But, we might not know what the right ε is!
- **How to compute ε-net for every possible ε?**
  - And do it all at once
  - Answer: ??



Intuition behind Greedy Permutation

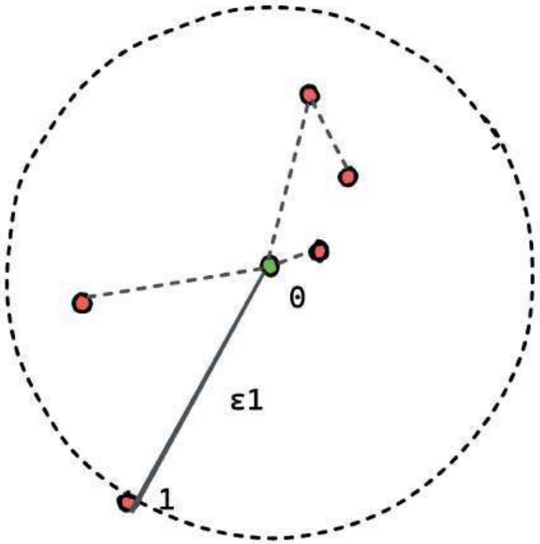
- Key insight:
  - If we have a **really really big ε**, one point would be enough (as it would cover everybody!)
  - For **really really small ε**, need to include all the points in the ε-net
- How does it change from one point to all of the points?
  - Start shrinking down the radius!

Step by step workout of Greedy Permutation

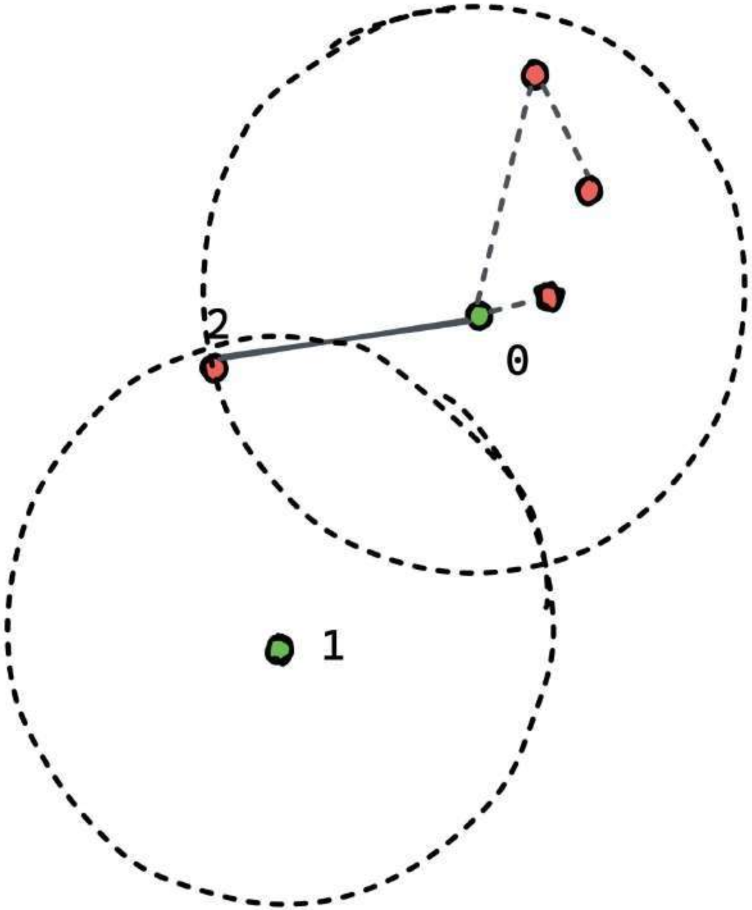
Let the set of all points be P
Initial point 0 is in set P0

Greedy Permutation: Every new point that we add is the next farthest point from the current set P to Pi

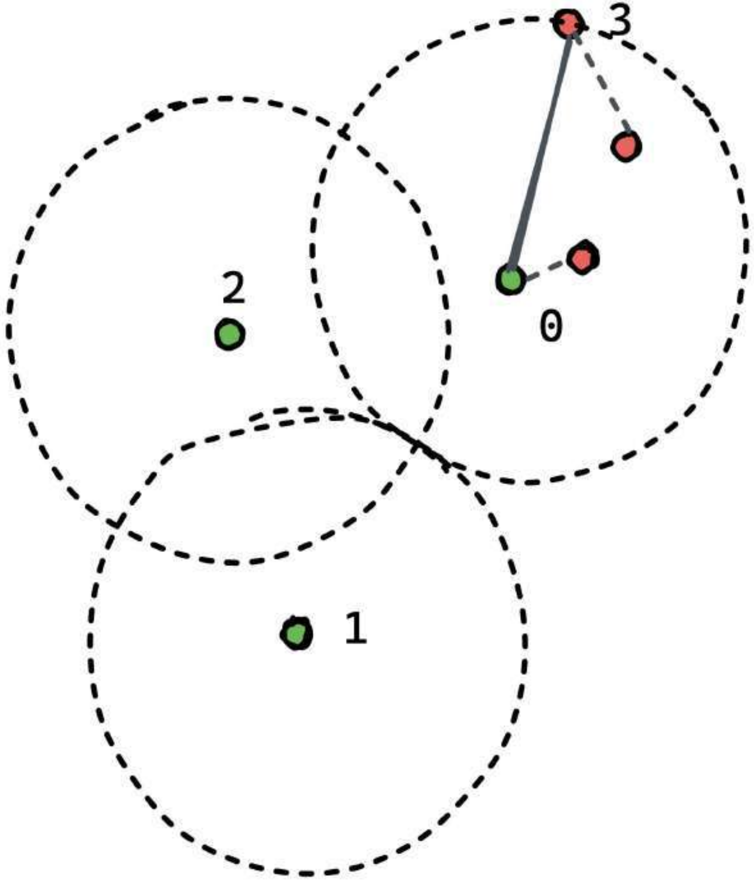
ε1 is the INSERTION DISTANCE OF 1



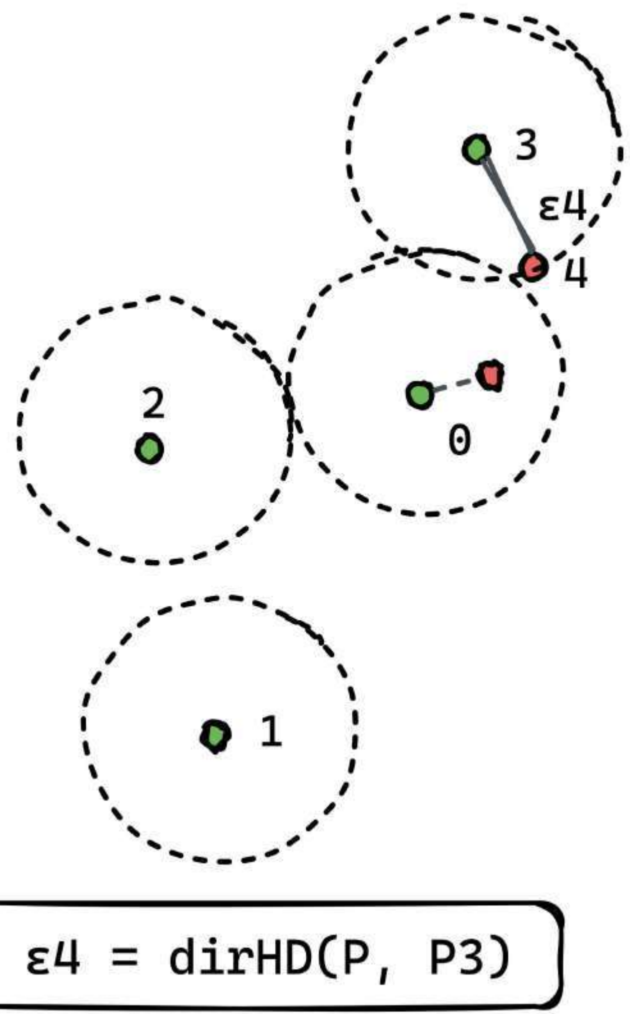
P1 = P0 U {1}



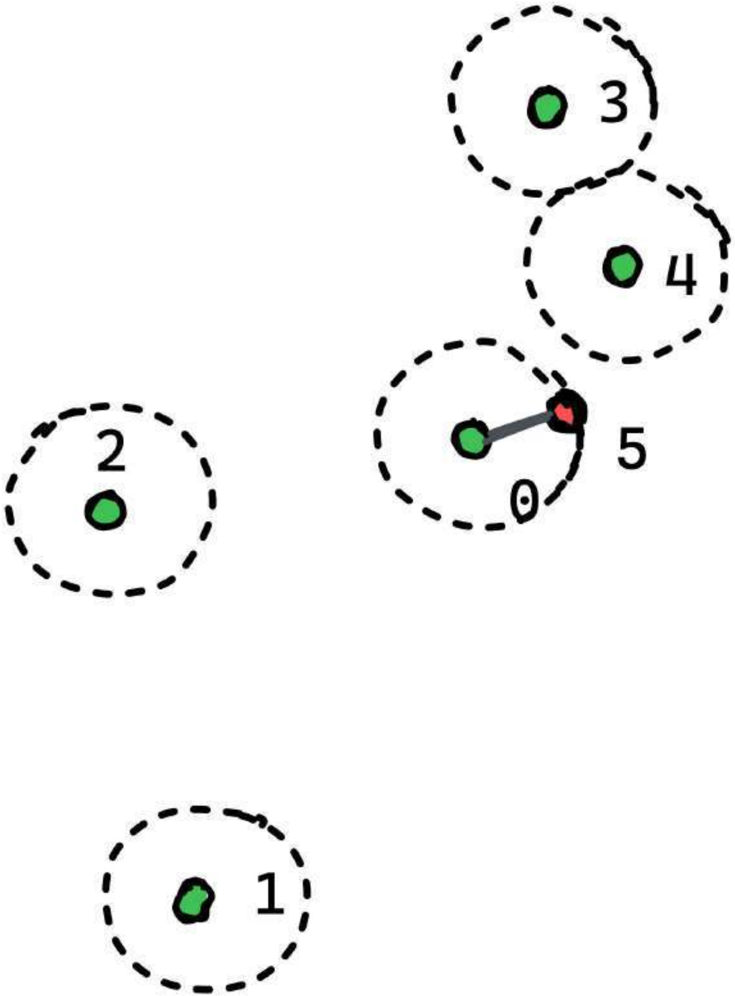
P2 = P1 U {2}

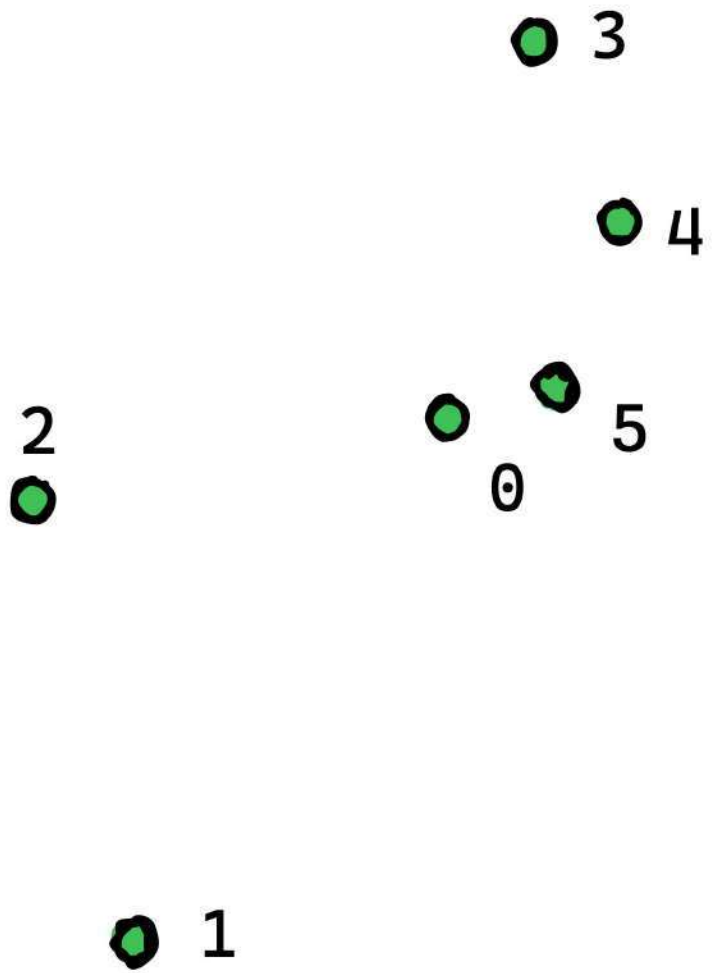


P3 = P2 U {3}



3
ε4
4
2
0
1
ε4 = dirHD(P, P3)


3
4
2
5
0
1



NOTE:

1. It often suffices to have an α-APPROXIMATE GREEDY PERMUTATION.

2. Can be computed in O(nlogn) time (Har Peled and Mendel)

3. We define: predecessor mapping T

It maps each point in P to its closest point in the prefix Pi

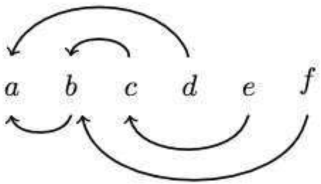
4. ε1 ≥ ε2 ≥ ε5

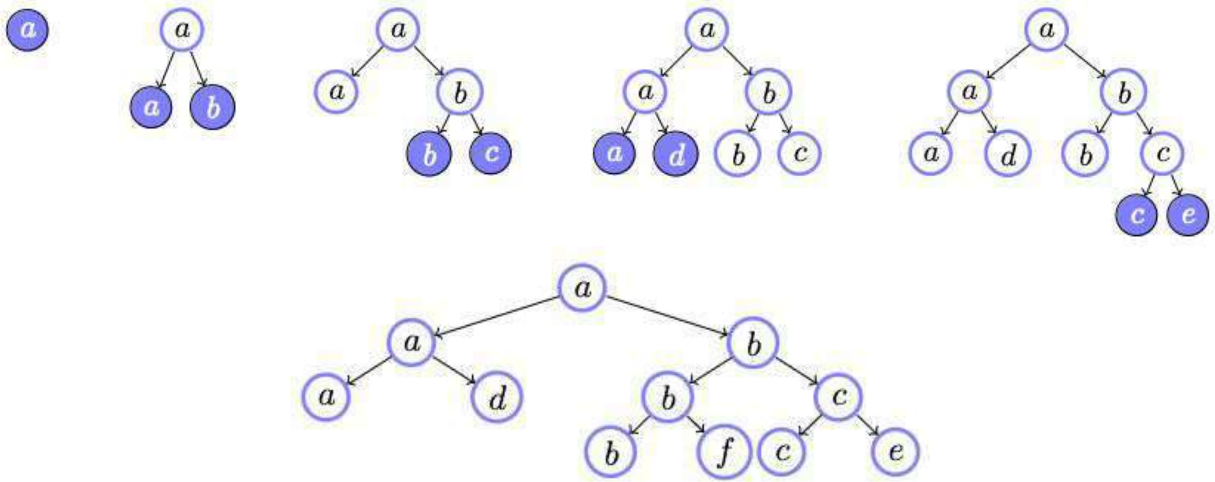
5. Pi is the εi-net

Greedy Trees

Construction of a Greedy Tree

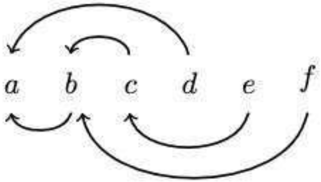
- Greedy Tree? **Binary tree that uses Greedy Permutation to achieve packing guarantees**
- Given Greedy Permutation (`G`) and Predecessor mapping (`T`) for point set `P`
- Radius of Node?

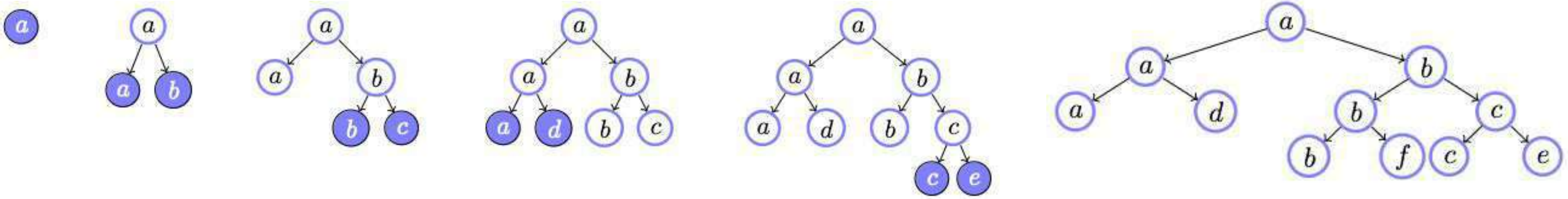




Construction of a Greedy Tree

- Greedy Tree? **Binary tree that uses Greedy Permutation to achieve packing guarantees**
- Given Greedy Permutation (G) and Predecessor mapping (T) for point set P
- Incremental construction:
  - Start with a root node centered at p_0
  - Iterate through the points b in the permutation (starting at p_1)
    - Let $a = T(b)$ and let x be the unique leaf of G centered at a
    - Create new nodes centered at a and b and assign them to be the left and right children of x





What kind of a binary tree is Greedy Tree?

- **Ball Tree**
- A ball tree on a set `A` is a binary tree defined by recursively partitioning `A`
  - Each node of the tree stores a **center** and a **radius** that covers the points in its leaves

Motivation behind Greedy Tree

- Lack of strong theoretical guarantees for ball trees
  - Leading to complicated data structures such as cover trees and net-trees
- Greedy Tree *provides strong theoretical guarantees* for proximity search queries
  - **Constructed in `O(nlogn)` time using Greedy Permutation**
- The radius of a node p is bounded

$$r_p \leq \frac{\varepsilon_p}{\alpha - 1}.$$

- Packing guarantee:
  - Let `X` be a set of pairwise independent nodes from a Greedy Tree. Then the centers of `X` are:

$$\frac{(\alpha - 1)r}{\alpha} - \text{packed}$$

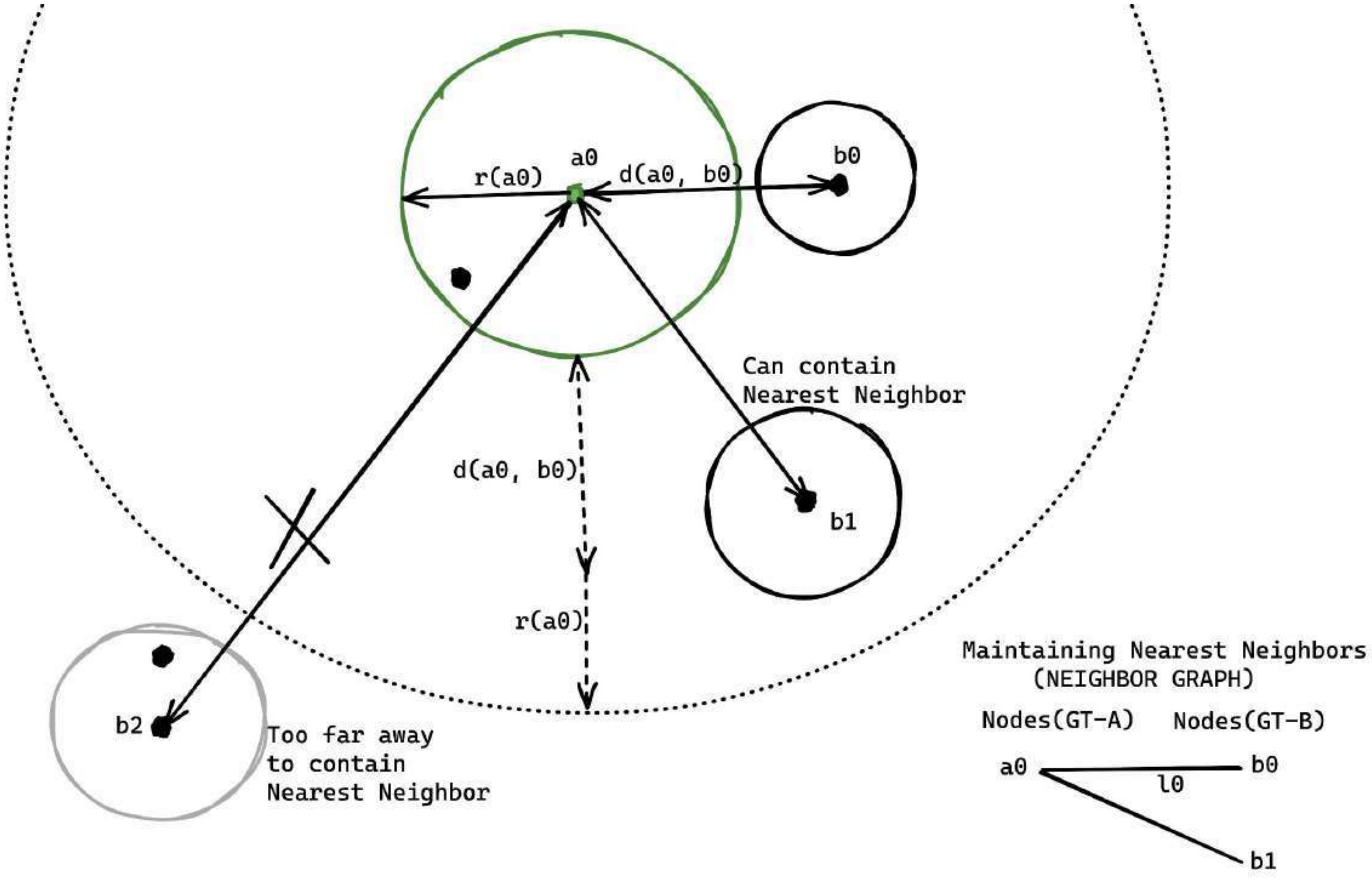
where `r` is the minimum radius of any parent of a node in `X`.

Hausdorff Distance using Greedy Trees

A High Level Glance

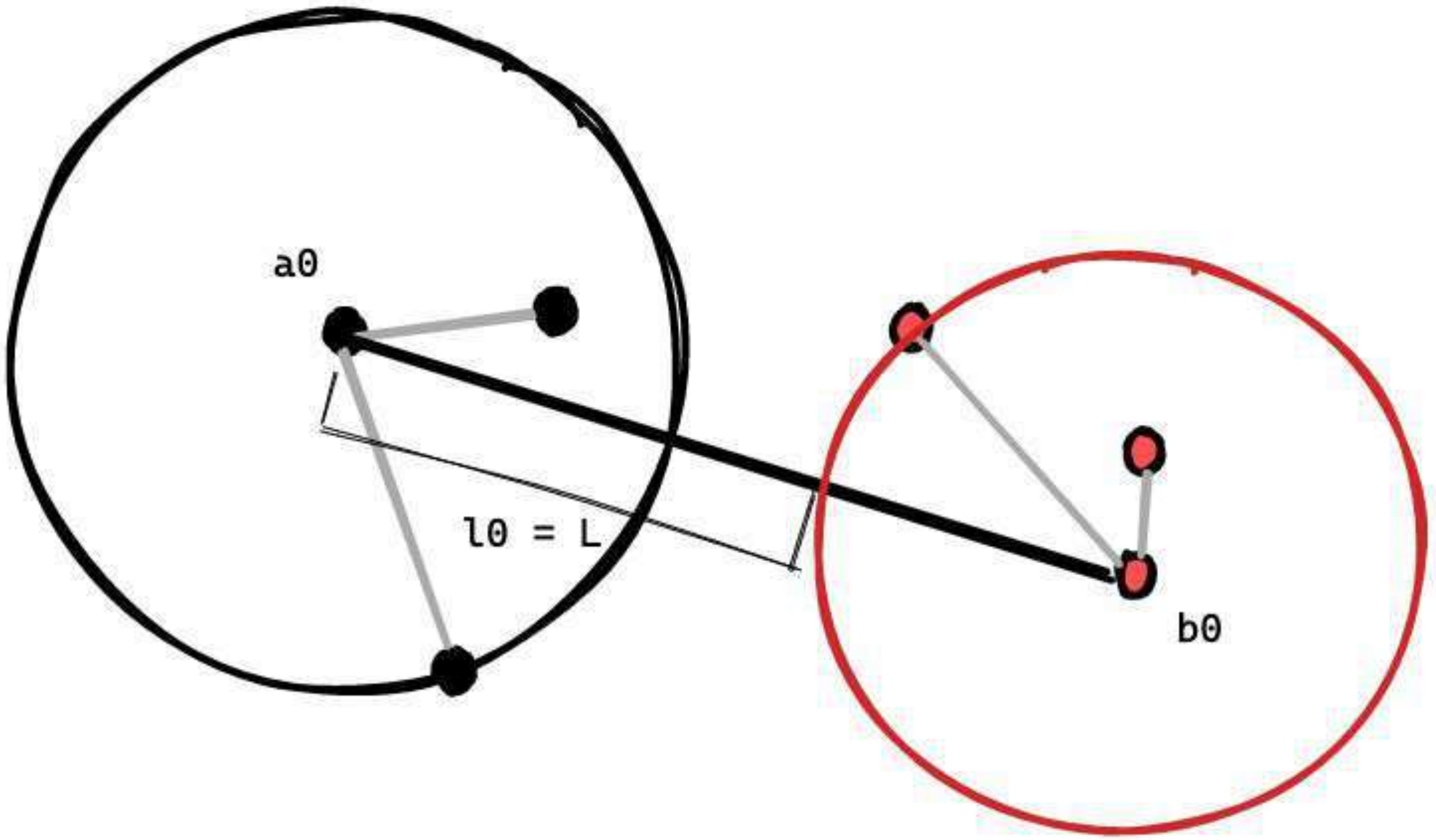
- What?
  - Computing (1+ε)-approximate **directed** Hausdorff Distance in linear time
    - with `O(n log n)` preprocessing time (for Greedy Permutation)
- Input
  - Pair of Greedy Trees G_A and G_B for sets A and B, respectively
  - List of their nodes in non-increasing order of radius
    - Iteration order of the algorithm
  - Approximation Factor ε

Intuition behind the Algorithm

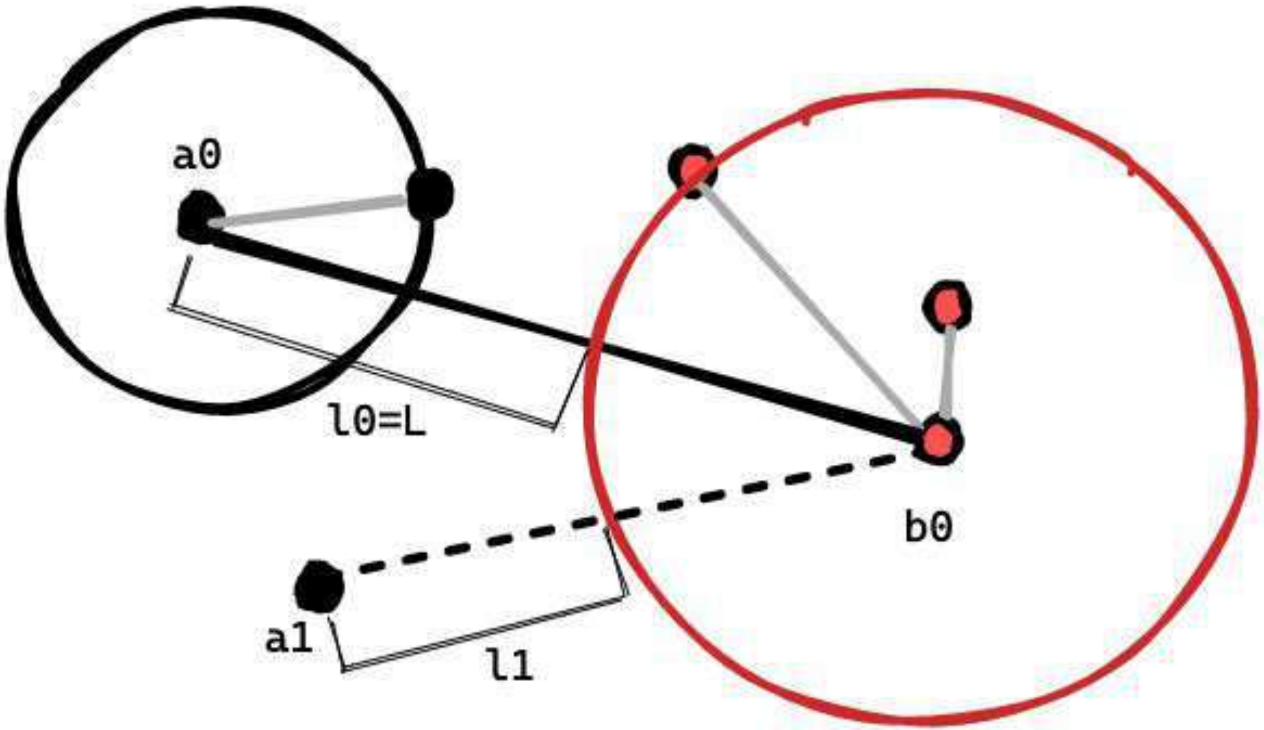


Step by step workout of the Algorithm

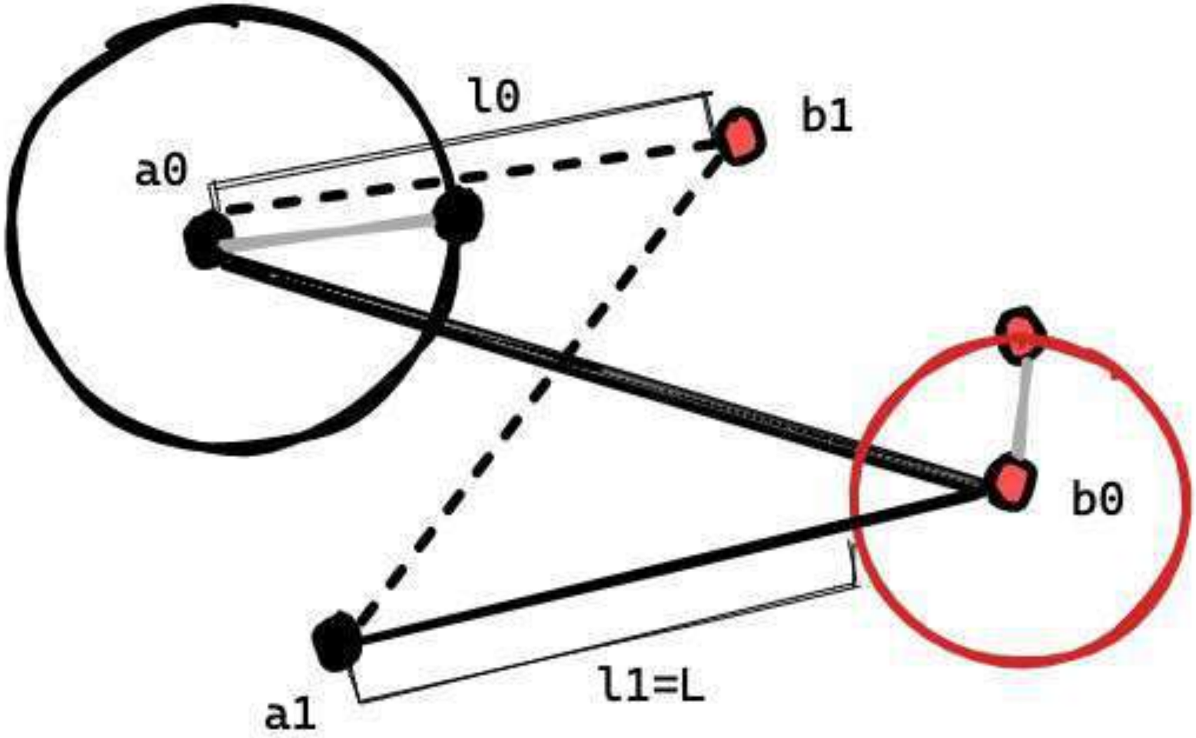
Initializing the Neighbor Graph



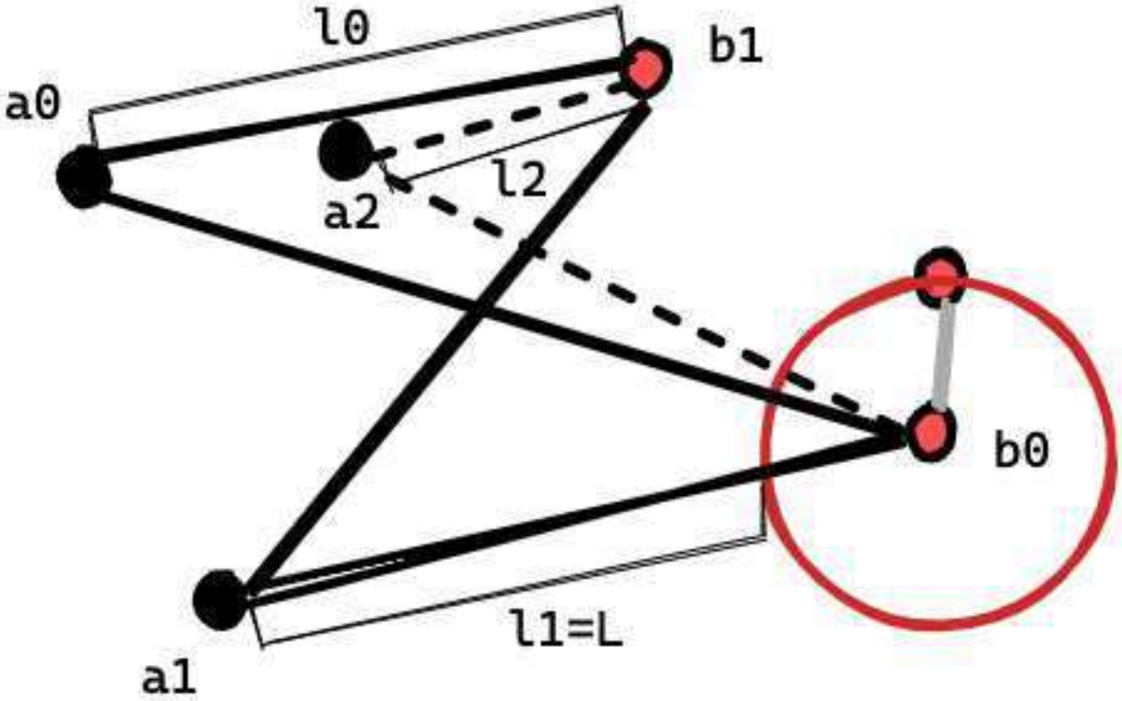
Case: Node in GT-A



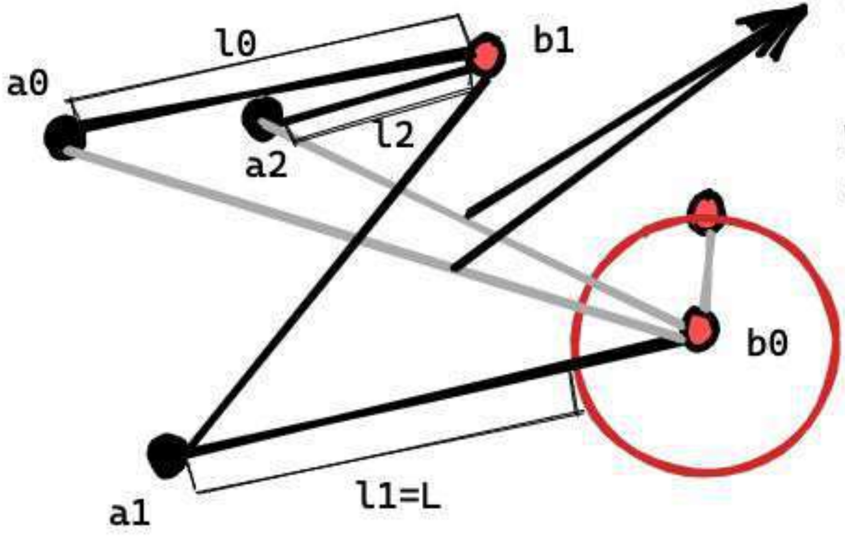
Case: Node in GT-B



Case: Node in GT-A



Case: Pruning

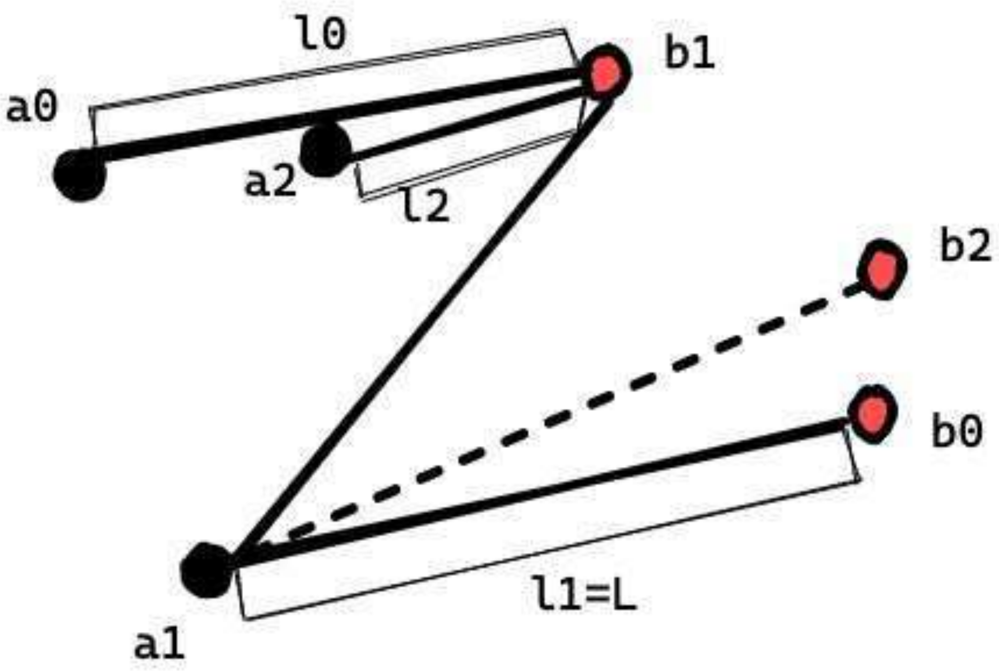


Pruning edges
(a0, b0) and
(a2, b0)

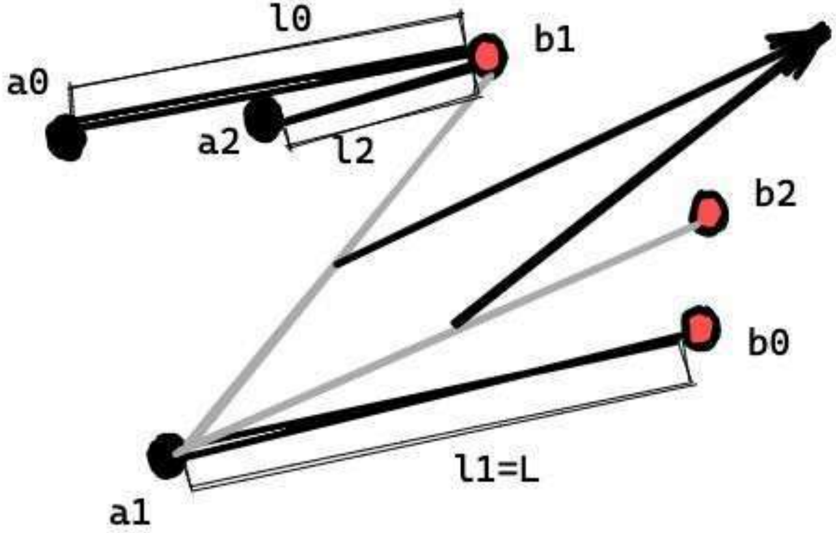
WHY? For (a0, b0):

$d(a0, b1) + r(a0) < d(a0, b0) - r(a0) - r(b0)$

Case: Node in GT-B



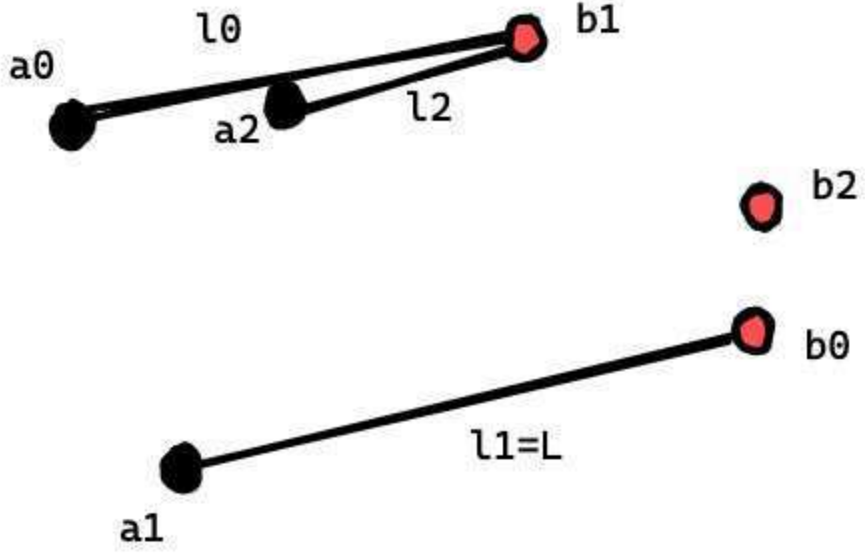
Case: Pruning



Pruning edges
(a1, b1) and
(a1, b2)

WHY? For (a1, b1):
$d(a1, b0) + r(a1) < d(a1, b1) - r(a1) - r(b1)$

Case: Hausdorff Distance



DIRECTED HAUSDORFF DISTANCE FROM A TO B
IS d(a1, b0) = l0 = L

A better Stopping Criteria

- Key Idea: Stop once the unprocessed nodes have radii **too small** to significantly affect the LB, and return L
- What is “too small”?
    - If the radius r of the largest node yet to be processed is

$$r \leq \frac{\varepsilon}{2} L$$

Correctness

- We show that:
  - $L \leq \mathrm{dirHD}(A, B) \leq L + 2r$
- Thus, when $r \leq \frac{\varepsilon}{2}L$
  - Directed HD is $(1+\varepsilon)$-approximate

Conclusion

- Questions?
- Code is available https://github.com/donsheehy/greedypermutation
- *Thank you Don, Siddharth, Oliver, and Kirk*

Additional Resources

- Proximity Search in the Greedy Tree
  - *Symposium on Simplicity in Algorithms (SOSA), 2023*
- Linear-Time Approximate Hausdorff Distance
  - *The 30th Fall Workshop on Computational Geometry, 2022*
- Approximating the Directed Hausdorff Distance
  - Submitted to *35th Canadian Conference on Computational Geometry, 2023*
- Greedy Permutations and Finite Voronoi Diagrams
  - Submitted to Multimedia Exposition in Computational Geometry, SoCG 2023
- The Finite Voronoi Method
  - Prepared for resubmission