Fast *k*-Nearest Neighbour Search via Prioritized DCI

Ke Li **Jitendra Malik**

{ke.li,malik}@eecs.berkeley.edu

Berkeley UNIVERSITY OF CALIFORNIA

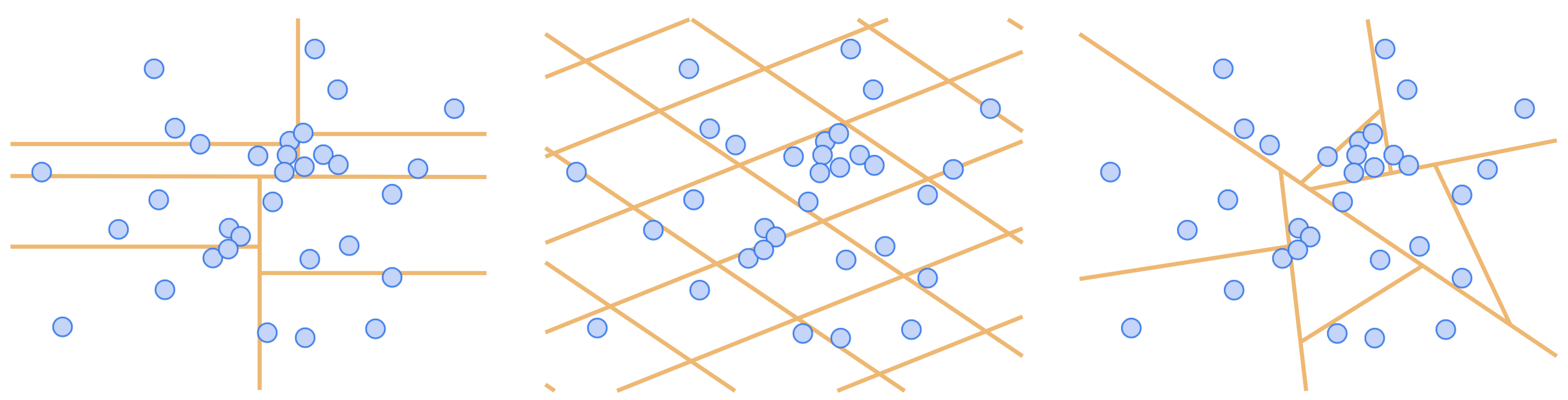
Introduction

- Existing methods suffer from the curse of dimensionality.
- Most existing methods rely on a divide-and-conquer strategy known as space partitioning.
- We present a new algorithm that overcomes the curse of dimensionality, which has:
  - Time complexity: linear in ambient dimensionality, sub-linear in intrinsic dimensionality and size of the dataset.
  - Space complexity: independent of ambient dimensionality and linear in size of the dataset.

The Case Against Space Partitioning

- Most existing methods rely on space partitioning:

k-d tree | LSH | RP tree



- Problems:
  - As dimensionality increases, volume of space grows exponentially ⇒ either the number or the size of cells must grow exponentially.
  - "Field of view" is limited to the cell containing the query; algorithm unaware of points in adjacent cells.
  - As dimensionality increases, surface area grows faster than volume ⇒ points likely to be near cell boundaries.
  - Choosing good partitioning is non-trivial. Once chosen, cannot adapt to changes in data density.

Algorithm

Algorithm 1 Data structure construction procedure

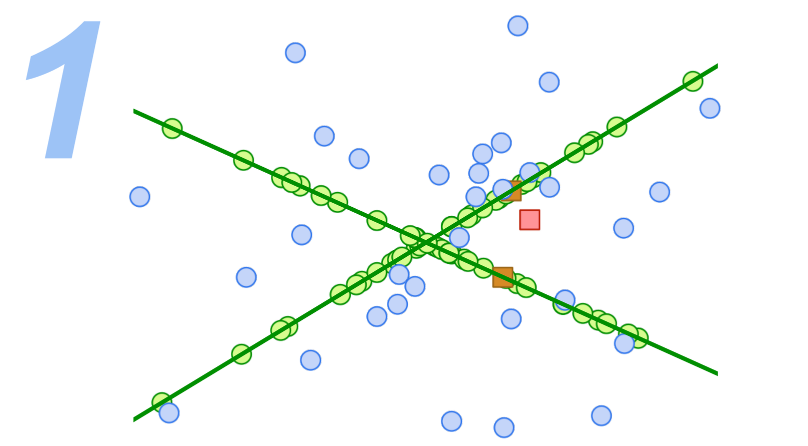
Require: A dataset D of n points $p^1, \ldots, p^n$, the number of simple indices m that constitute a composite index and the number of composite indices L

```
function CONSTRUCT(D, m, L)
    {u_jl}_{j∈[m],l∈[L]} ← mL random unit vectors in R^d
    {T_jl}_{j∈[m],l∈[L]} ← mL empty binary search trees or skip lists
    for j = 1 to m do
        for l = 1 to L do
            for i = 1 to n do
                p̄^i_jl ← ⟨p^i, u_jl⟩
                Insert (p̄^i_jl, i) into T_jl with p̄^i_jl being the key and i being the value
            end for
        end for
    end for
    return {(T_jl, u_jl)}_{j∈[m],l∈[L]}
end function
```

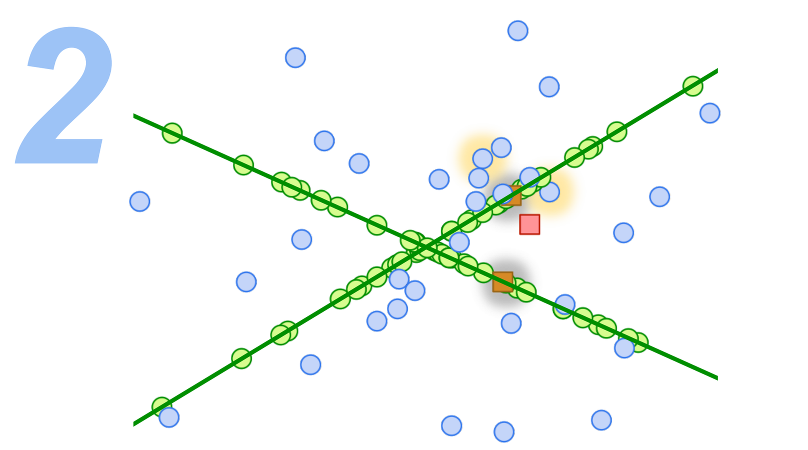
Algorithm 2 k-nearest neighbour querying procedure

Require: Query point q in $\mathbb{R}^d$, binary search trees/skip lists and their associated projection vectors $\{(T_{jl}, u_{jl})\}_{j\in[m],l\in[L]}$, the number of points to retrieve k_0 and the number of points to visit k_1 in each composite index

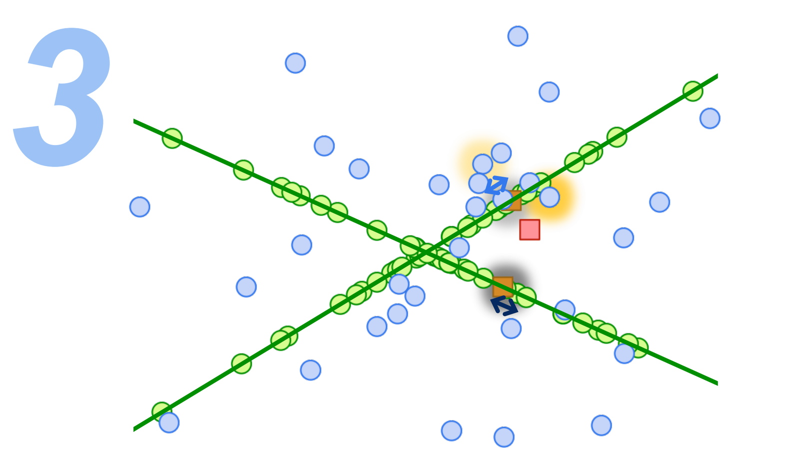
```
function QUERY(q, {(T_jl, u_jl)}_{j,l}, k_0, k_1)
    C_l ← array of size n with entries initialized to 0  ∀l ∈ [L]
    q̄_jl ← ⟨q, u_jl⟩ ∀j ∈ [m], l ∈ [L]
    S_l ← ∅ ∀l ∈ [L]
    P_l ← empty priority queue  ∀l ∈ [L]
    for l = 1 to L do
        for j = 1 to m do
            (p̄^(1)_jl, h^(1)_jl) ← the node in T_jl whose key is the closest to q̄_jl
            Insert (p̄^(1)_jl, h^(1)_jl) with priority −|p̄^(1)_jl − q̄_jl| into P_l
        end for
    end for
    for i' = 1 to k_1 − 1 do
        for l = 1 to L do
            if |S_l| < k_0 then
                (p̄^(i)_jl, h^(i)_jl) ← the node with the highest priority in P_l
                Remove (p̄^(i)_jl, h^(i)_jl) from P_l and insert the node in T_jl whose key is the next closest to q̄_jl,
                    which is denoted as (p̄^(i+1)_jl, h^(i+1)_jl), with priority −|p̄^(i+1)_jl − q̄_jl| into P_l
                C_l[h^(i)_jl] ← C_l[h^(i)_jl] + 1
                if C_l[h^(i)_jl] = m then
                    S_l ← S_l ∪ {h^(i)_jl}
                end if
            end if
        end for
    end for
    return k points in ⋃_{l∈[L]} S_l that are the closest in Euclidean distance in R^d to q
end function
```



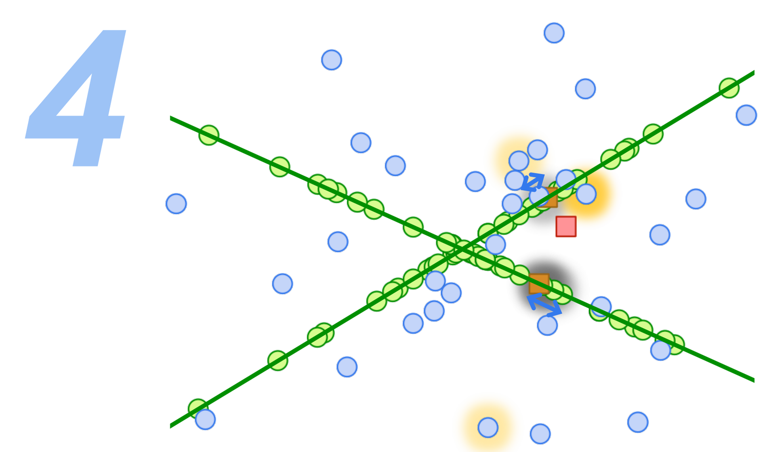
Project data points and query along random directions.



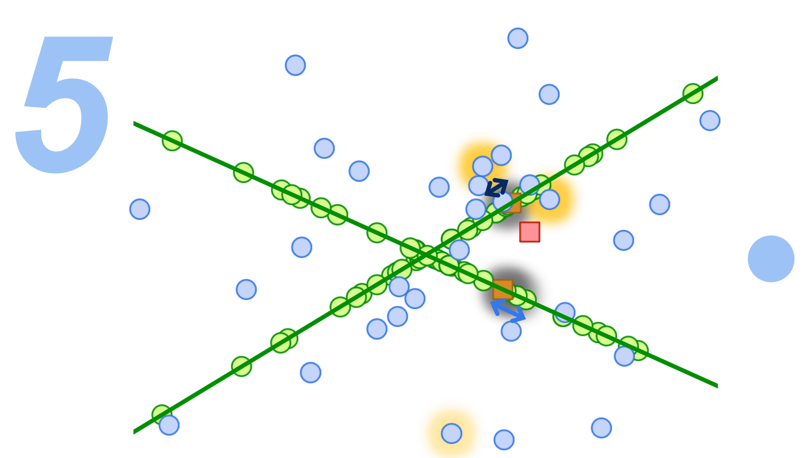
Add the closest point to the query along each projection direction to the frontier.



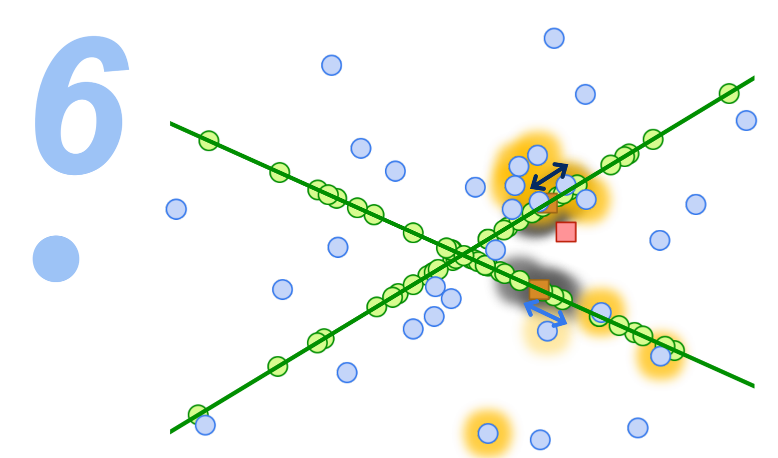
Visit the point on the frontier with the shortest projected distance to the query.



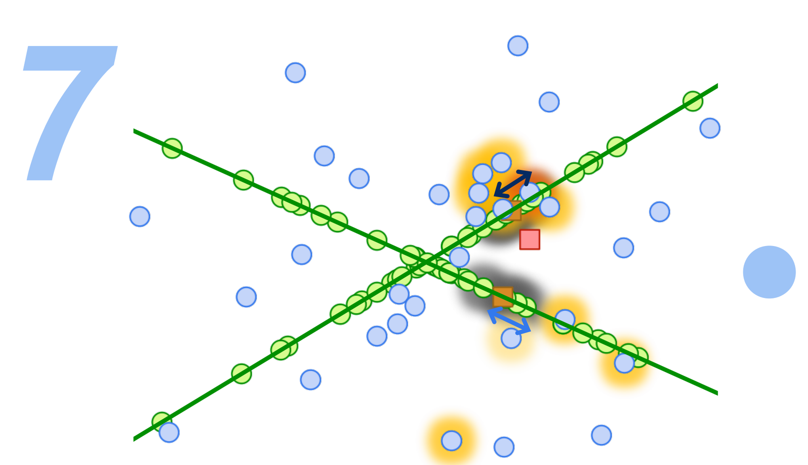
Add the next closest point along the most recently processed direction to the frontier.



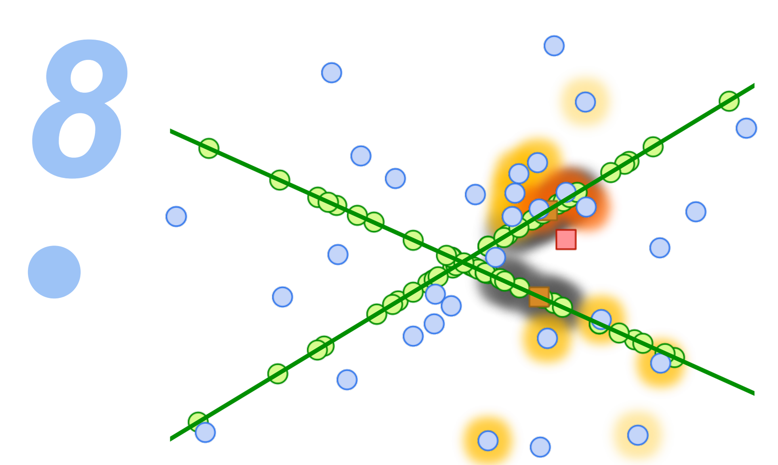
Visit the point on the frontier with the shortest projected distance to the query.



Points highlighted in dark orange have been visited; visit the next point on the frontier.



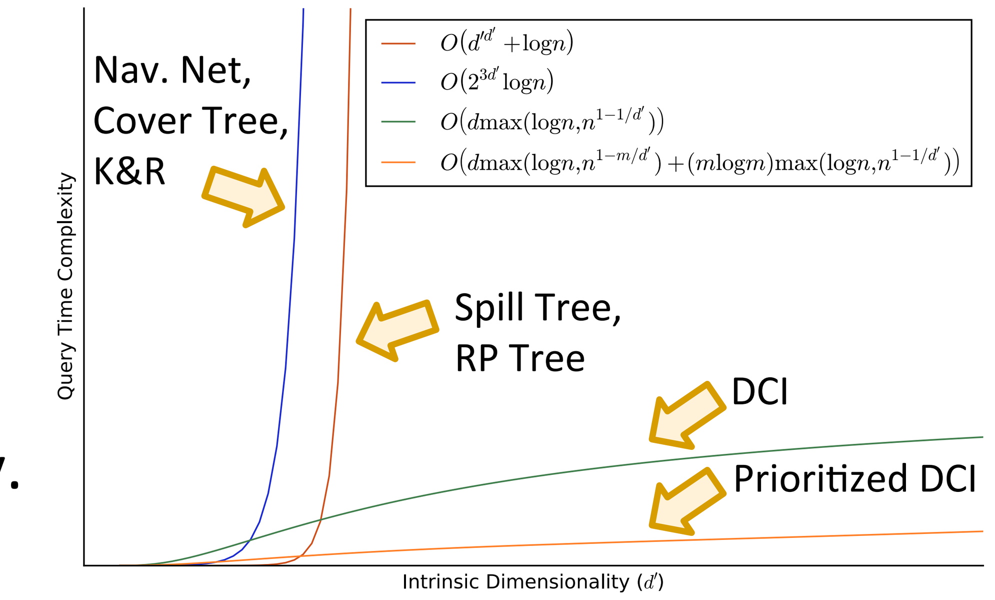
Current point has now been visited along all directions and is added to candidate set.



Search exhaustively over all points in the candidate set and return the k closest ones.

Complexity

- Key Observation:
  - Points are added to the candidate set in the order of maximum projected distance to the query.
  - Maximum projected distance is a lower bound of the true distance.
  - As # of projection directions → ∞, this → true distance.

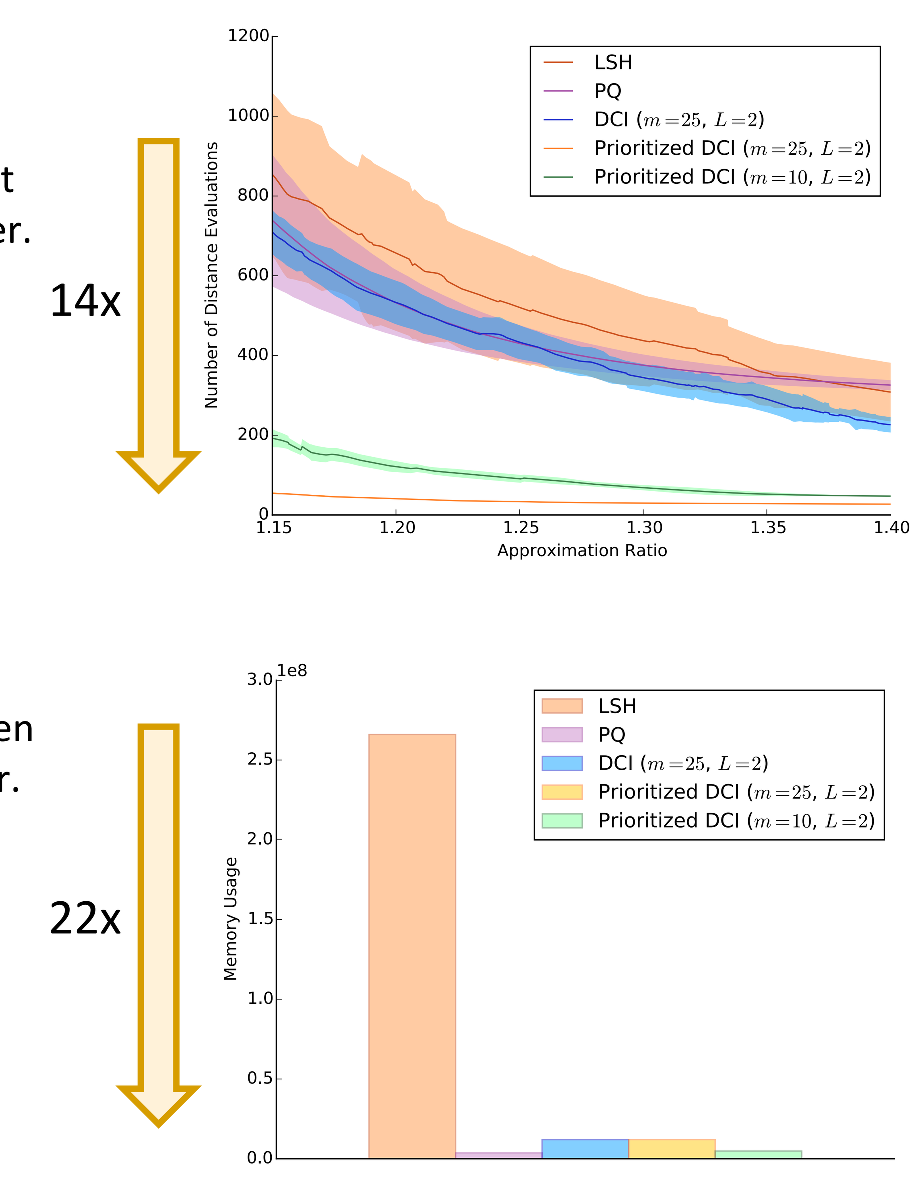


Construction	$O(m(dn + n\log n))$
Query	$O(dk\max(\log(n/k), (n/k)^{1-m/d'}) + mk\log m(\max(\log(n/k), (n/k)^{1-1/d'}))$
Insertion	$O(m(d + \log n))$
Deletion	$O(m\log n)$
Space	$O(mn)$

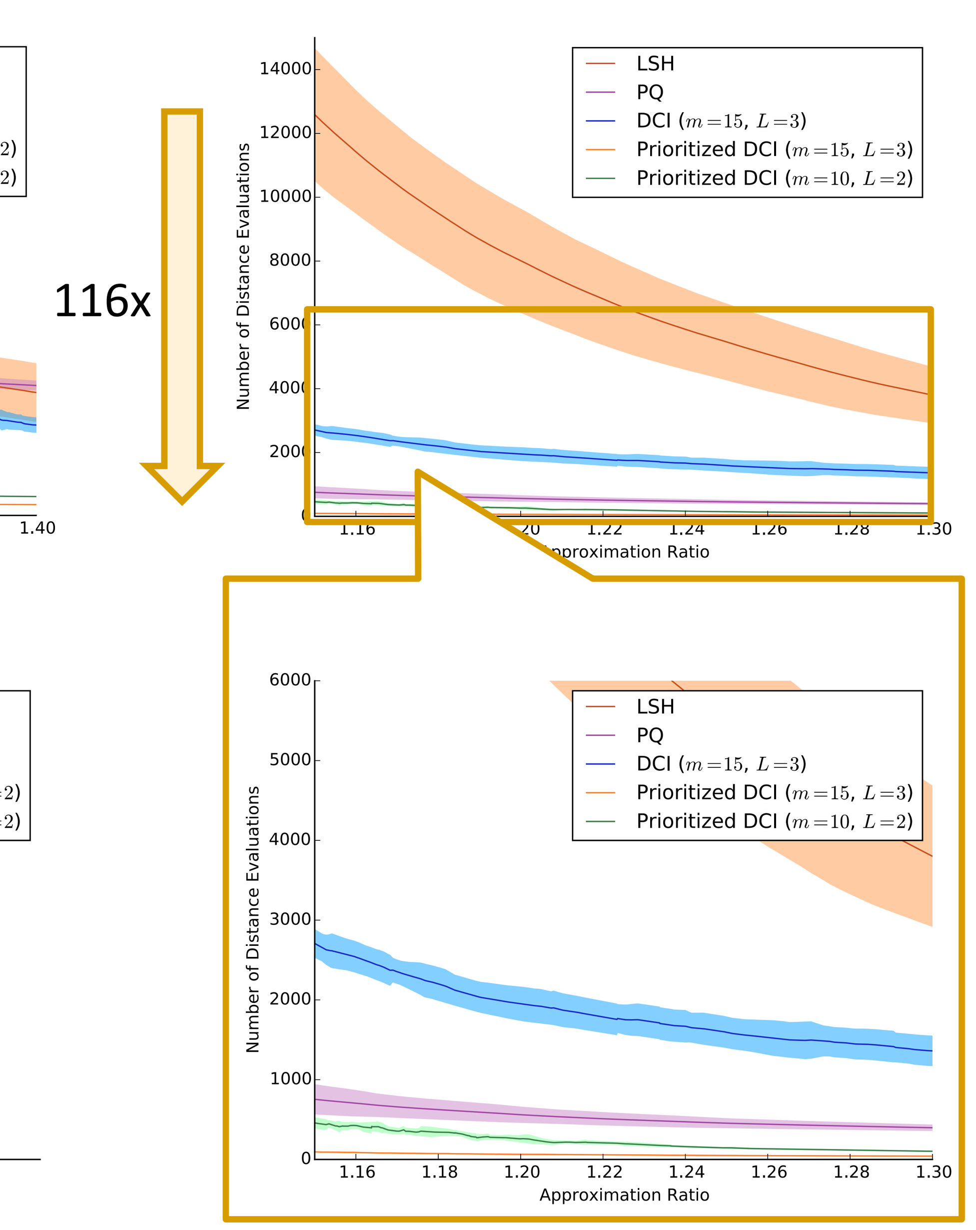
where $m \geq 1$ is # of projection directions

Experiments

CIFAR-100



MNIST



$$\text{approximation ratio} = \frac{\text{distance to retrieved nearest neighbours}}{\text{distance to true nearest neighbours}}$$