Jack Spalding-Jamieson (Jack S-J)
jacksj@uwaterloo.ca

Independent

Scalable k-Means Clustering for Large k via Seeded Approximate Nearest-Neighbor Search

Joint work with Eliot Robson and Da Wei Zheng

k-Means/Sum of Squares Clustering: Quick Review

Input

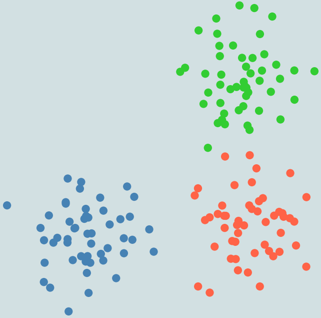
- k: a parameter (e.g., $k = 3$)
- n d-dimensional vectors

k-Means/Sum of Squares Clustering: Quick Review

Input

- k: a parameter (e.g., $k = 3$)
- n d-dimensional vectors

Output



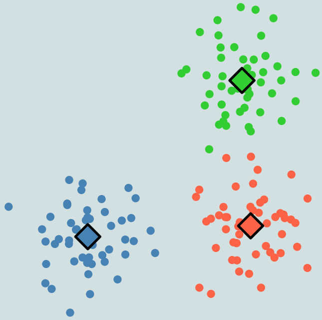
- Clusters $\{C_i\}_{i \in [k]}$ with centroids $\{\mu_i\}_{i \in [k]}$
- Objective: $\min \sum_{i=1}^{k} \sum_{x,y \in C_i} \|x - y\|^2$

k-Means/Sum of Squares Clustering: Quick Review

Input

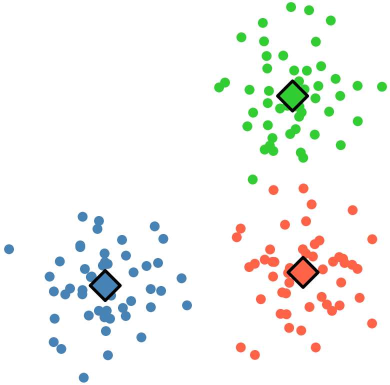
- k: a parameter (e.g., $k = 3$)
- n d-dimensional vectors

Output



- Clusters $\{C_i\}_{i\in[k]}$ with centroids $\{\mu_i\}_{i\in[k]}$
- **Objective**: $\min \sum_{\mathbf{i=1}}^{\mathbf{k}} \sum_{\mathbf{x}\in \mathbf{C_i}} \|\mathbf{x} - \mu_{\mathbf{i}}\|^{\mathbf{2}}$
(Equivalent by Huygens' theorem)

k-means is NP-hard, even for $k = 2$.[1]



[1] Aloise et al., *NP-hardness of Euclidean sum-of-squares clustering*

[2] Arthur & Vassilvitskii, *k-means++: The Advantages of Careful Seeding*

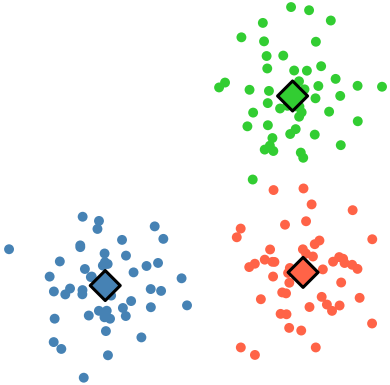
[3] Bahmani et al., *Scalable k-means++*

[4] Lloyd, *Least square quantization in PCM*

[5] Bahmani's slides on `k-means||`

k-means is NP-hard, even for $k = 2$.[1]

Two broad approaches for good (practical) solutions:



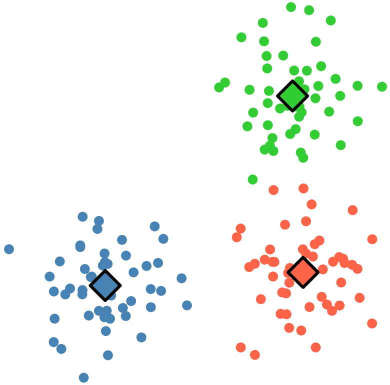
[1] Aloise et al., *NP-hardness of Euclidean sum-of-squares clustering*
[2] Arthur & Vassilvitskii, *k-means++: The Advantages of Careful Seeding*
[3] Bahmani et al., *Scalable k-means++*
[4] Lloyd, *Least square quantization in PCM*
[5] Bahmani's slides on `k-means||`

k-Means/Sum of Squares Clustering: Complexity and Approaches

k-means is NP-hard, even for $k = 2$.[1]

Two broad approaches for good (practical) solutions:

- **Local search: Lloyd's algorithm**[4]
  - Time Complexity per iteration: $O(nkd)$



[1] Aloise et al., *NP-hardness of Euclidean sum-of-squares clustering*

[2] Arthur & Vassilvitskii, *k-means++: The Advantages of Careful Seeding*

[3] Bahmani et al., *Scalable k-means++*

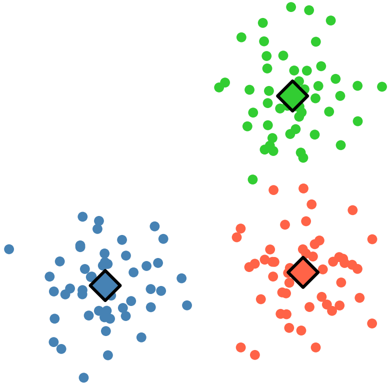
[4] Lloyd, *Least square quantization in PCM*

[5] Bahmani's slides on `k-means||`

k-means is NP-hard, even for $k = 2$.[1]

Two broad approaches for good (practical) solutions:

- **Local search:** Lloyd's algorithm[4]
  - Time Complexity per iteration: $O(nkd)$
- **Approximation algorithms:**



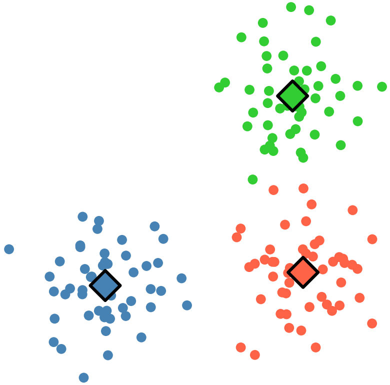
[1] Aloise et al., *NP-hardness of Euclidean sum-of-squares clustering*
[2] Arthur & Vassilvitskii, *k-means++: The Advantages of Careful Seeding*
[3] Bahmani et al., *Scalable k-means++*
[4] Lloyd, *Least square quantization in PCM*
[5] Bahmani's slides on `k-means||`

k-Means/Sum of Squares Clustering: Complexity and Approaches

k-means is NP-hard, even for $k = 2$.[1]

Two broad approaches for good (practical) solutions:

- **Local search:** Lloyd's algorithm[4]
  - Time Complexity per iteration: $O(nkd)$
- **Approximation algorithms:**
  - `k-means++`[2]
    - $O(nkd)$ time
    - $8(\ln k + 2)$ expected approximation



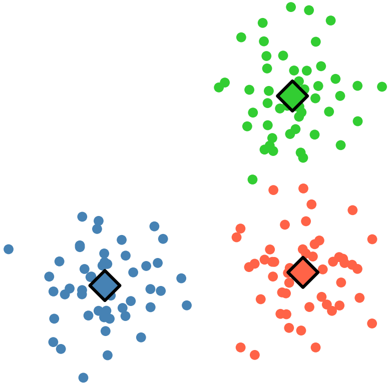
[1] Aloise et al., *NP-hardness of Euclidean sum-of-squares clustering*
[2] Arthur & Vassilvitskii, *k-means++: The Advantages of Careful Seeding*
[3] Bahmani et al., *Scalable k-means++*
[4] Lloyd, *Least square quantization in PCM*
[5] Bahmani's slides on `k-means||`

k-Means/Sum of Squares Clustering: Complexity and Approaches

k-means is NP-hard, even for $k = 2$.[1]

Two broad approaches for good (practical) solutions:

- **Local search:** Lloyd's algorithm[4]
  - Time Complexity per iteration: $O(nkd)$
- **Approximation algorithms:**
  - `k-means++`[2]
    - $O(nkd)$ time
    - $8(\ln k + 2)$ expected approximation
  - `k-means||`[3,5]
    - $O(c_1 \cdot nd + O(c_2 \cdot k^2 d)$ time, c_1, c_2 small in practice
    - $O(\log k)$ expected approximation



[1] Aloise et al., *NP-hardness of Euclidean sum-of-squares clustering*
[2] Arthur & Vassilvitskii, *k-means++: The Advantages of Careful Seeding*
[3] Bahmani et al., *Scalable k-means++*
[4] Lloyd, *Least square quantization in PCM*
[5] Bahmani's slides on `k-means||`

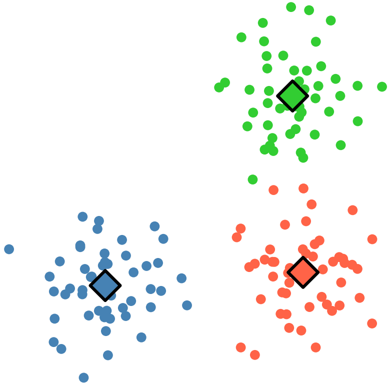
k-Means/Sum of Squares Clustering: Complexity and Approaches

k-means is NP-hard, even for $k = 2$.[1]

Two broad approaches for good (practical) solutions:

- **Local search:** Lloyd's algorithm[4]
  - Time Complexity per iteration: $O(nkd)$
- **Approximation algorithms:**
  - `k-means++`[2]
    - $O(nkd)$ time
    - $8(\ln k + 2)$ expected approximation
  - `k-means||`[3,5]
    - $O(c_1 \cdot nd + O(c_2 \cdot k^2 d)$ time, c_1, c_2 small in practice
    - $O(\log k)$ expected approximation

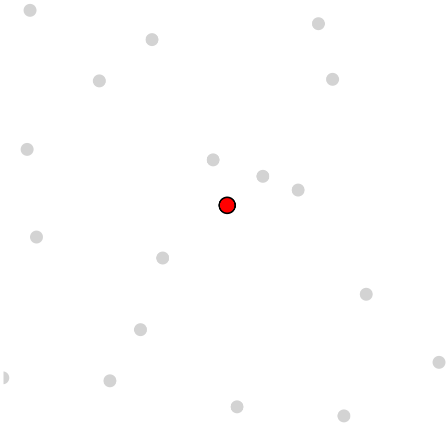
Better approximations known, not used in practice.



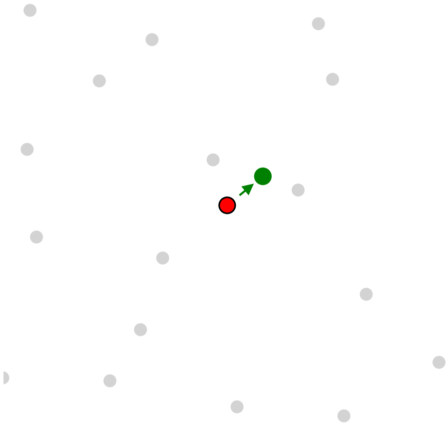
[1] Aloise et al., *NP-hardness of Euclidean sum-of-squares clustering*
[2] Arthur & Vassilvitskii, *k-means++: The Advantages of Careful Seeding*
[3] Bahmani et al., *Scalable k-means++*
[4] Lloyd, *Least square quantization in PCM*
[5] Bahmani's slides on `k-means||`

Nearest-Neighbor Search

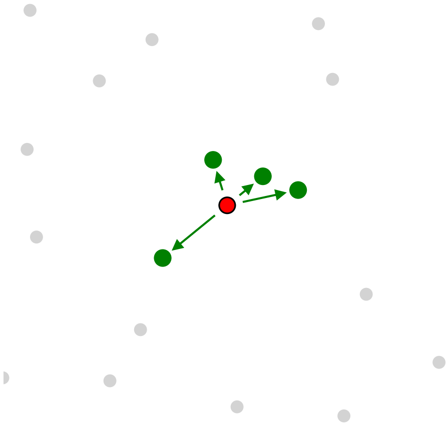
Nearest-Neighbor Search



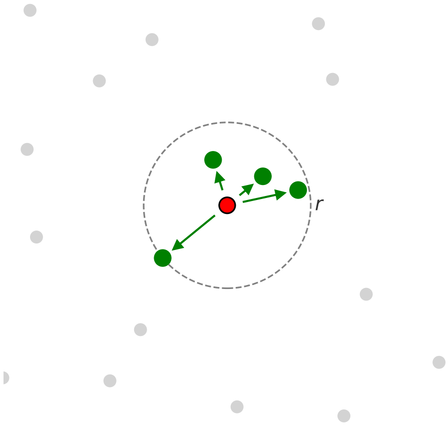
Nearest-Neighbor Search



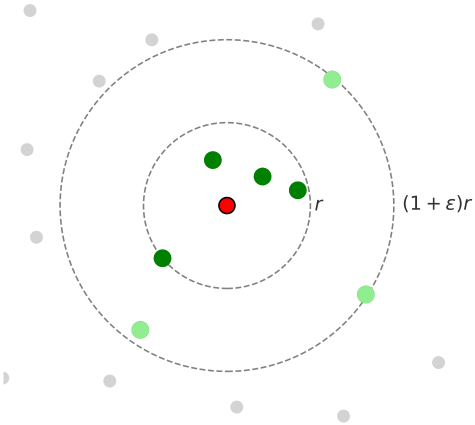
k-Nearest-Neighbor Search



k-Nearest-Neighbor Search

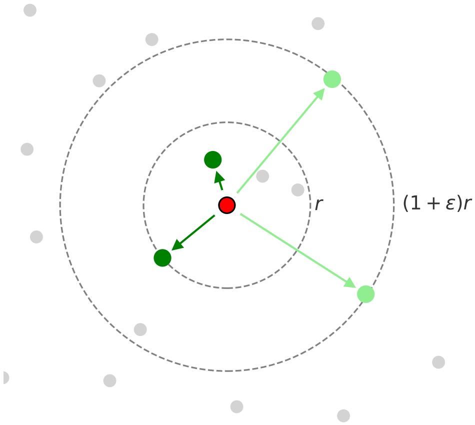


Approximate k-Nearest-Neighbor Search



Approximation factor: $(1 + \varepsilon)$

Approximate k-Nearest-Neighbor Search



Approximation factor: $(1 + \varepsilon)$
Recall: $\frac{2}{4} = 0.5$

Dataset sizes vs Existing Approaches

ANNS has three broad families of practical approaches:

- Quantization
- Space-partitioning/Clustering
- Search-graphs

Quantization is normally used alongside the other two.

Dataset sizes vs Existing Approaches

ANNS has three broad families of practical approaches:

- Quantization
- Space-partitioning/Clustering
- Search-graphs

Quantization is normally used alongside the other two.

Important fact: Search-graph approaches are (almost universally) best,

Dataset sizes vs Existing Approaches

ANNS has three broad families of practical approaches:

- Quantization
- Space-partitioning/Clustering
- Search-graphs

Quantization is normally used alongside the other two.

Important fact: Search-graph approaches are (almost universally) best,
but **only work if your data or quantized data fits in RAM** (prohibitively slow otherwise).

Dataset sizes vs Existing Approaches

ANNS has three broad families of practical approaches:

- Quantization
- Space-partitioning/Clustering
- Search-graphs

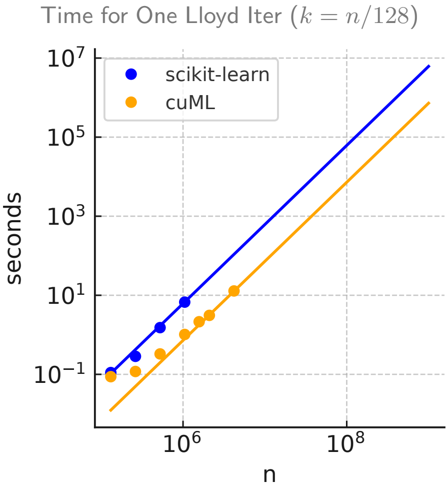
Quantization is normally used alongside the other two.

Important fact: Search-graph approaches are (almost universally) best,
but **only work if your data or quantized data fits in RAM** (prohibitively slow otherwise).

e.g. 200-dimensional f32 dataset with $20,000,000$ points requires 16GB (plus data structure size).

Some Motivation

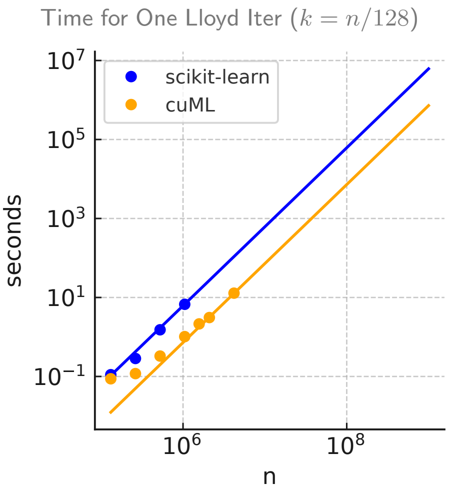
"Vector Similarity Search" is very popular in machine learning recently, with a lot of active development.



[1] Sivic & Zisserman, *Video Google: A text retrieval approach to object matching in videos*
[2] Jégou et al., *Product Quantization for Nearest Neighbor Search*
[3] Chen et al., *SPANN: Highly-Efficient Billion-scale Approximate Nearest Neighbor Search*
[4] Sun et al., *SOAR: Improved Indexing for Approximate Nearest Neighbor Search*

"Vector Similarity Search" is very popular in machine learning recently, with a lot of active development.

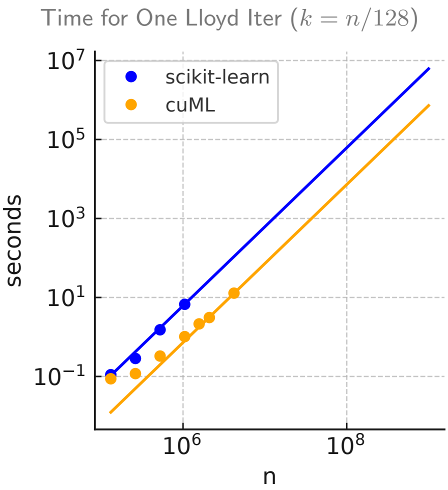
- **Many methods used in industry for massive datasets[1,2,3,4] solve k-means as a sub-routine, and would benefit from very large k.**



[1] Sivic & Zisserman, *Video Google: A text retrieval approach to object matching in videos*
[2] Jégou et al., *Product Quantization for Nearest Neighbor Search*
[3] Chen et al., *SPANN: Highly-Efficient Billion-scale Approximate Nearest Neighbor Search*
[4] Sun et al., *SOAR: Improved Indexing for Approximate Nearest Neighbor Search*

"Vector Similarity Search" is very popular in machine learning recently, with a lot of active development.

- Many methods used in industry for massive datasets[1,2,3,4] solve k-means as a sub-routine, and would benefit from very large k.
- **Many would benefit from large k ($k \approx n/c$ for small c), but current methods are too slow.**



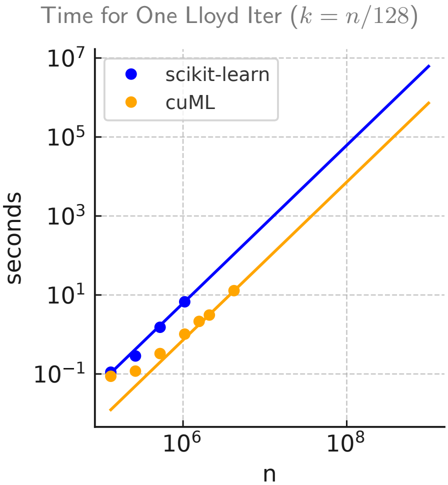
[1] Sivic & Zisserman, *Video Google: A text retrieval approach to object matching in videos*
[2] Jégou et al., *Product Quantization for Nearest Neighbor Search*
[3] Chen et al., *SPANN: Highly-Efficient Billion-scale Approximate Nearest Neighbor Search*
[4] Sun et al., *SOAR: Improved Indexing for Approximate Nearest Neighbor Search*

Some Motivation

"Vector Similarity Search" is very popular in machine learning recently, with a lot of active development.

- Many methods used in industry for massive datasets[1,2,3,4] solve k-means as a sub-routine, and would benefit from very large k.
- Many would benefit from large k ($k \approx n/c$ for small c), but current methods are too slow.

All practical methods take $\Omega(k^2)$ time.



[1] Sivic & Zisserman, *Video Google: A text retrieval approach to object matching in videos*
[2] Jégou et al., *Product Quantization for Nearest Neighbor Search*
[3] Chen et al., *SPANN: Highly-Efficient Billion-scale Approximate Nearest Neighbor Search*
[4] Sun et al., *SOAR: Improved Indexing for Approximate Nearest Neighbor Search*

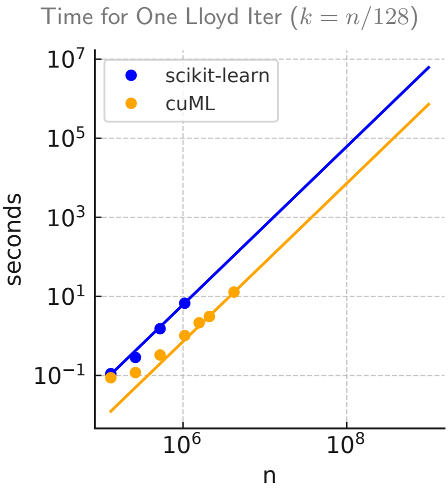
Some Motivation

"Vector Similarity Search" is very popular in machine learning recently, with a lot of active development.

- Many methods used in industry for massive datasets[1,2,3,4] solve k-means as a sub-routine, and would benefit from very large k.
- Many would benefit from large k ($k \approx n/c$ for small c), but current methods are too slow.

All practical methods take $\Omega(k^2)$ time.

Focus on large $n \in [10^6, 10^9]$, $k \approx n/c$, and $d \geq 100$.



[1] Sivic & Zisserman, *Video Google: A text retrieval approach to object matching in videos*
[2] Jégou et al., *Product Quantization for Nearest Neighbor Search*
[3] Chen et al., *SPANN: Highly-Efficient Billion-scale Approximate Nearest Neighbor Search*
[4] Sun et al., *SOAR: Improved Indexing for Approximate Nearest Neighbor Search*

Some Motivation

"Vector Similarity Search" is very popular in machine learning recently, with a lot of active development.

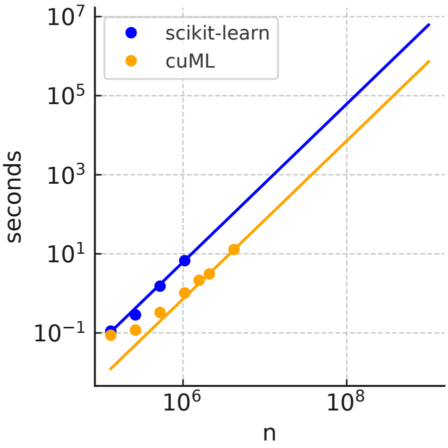
- Many methods used in industry for massive datasets[1,2,3,4] solve k-means as a sub-routine, and would benefit from very large k.
- Many would benefit from large k ($k \approx n/c$ for small c), but current methods are too slow.

All practical methods take $\Omega(k^2)$ time.

Focus on large $n \in [10^6, 10^9]$, $k \approx n/c$, and $d \geq 100$.

- **Days to weeks with current methods!** $\rightarrow$

Time for One Lloyd Iter ($k = n/128$)



[1] Sivic & Zisserman, *Video Google: A text retrieval approach to object matching in videos*
[2] Jégou et al., *Product Quantization for Nearest Neighbor Search*
[3] Chen et al., *SPANN: Highly-Efficient Billion-scale Approximate Nearest Neighbor Search*
[4] Sun et al., *SOAR: Improved Indexing for Approximate Nearest Neighbor Search*

"Vector Similarity Search" is very popular in machine learning recently, with a lot of active development.

- Many methods used in industry for massive datasets[1,2,3,4] solve k-means as a sub-routine, and would benefit from very large k.
- Many would benefit from large k ($k \approx n/c$ for small c), but current methods are too slow.

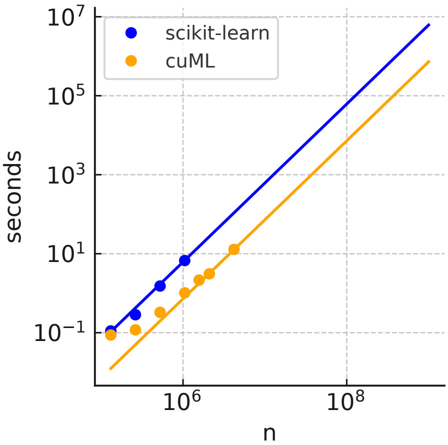
All practical methods take $\Omega(k^2)$ time.

Focus on large $n \in [10^6, 10^9]$, $k \approx n/c$, and $d \geq 100$.

- Days to weeks with current methods! $\rightarrow$

Focus on real performance, not asymptotics.

Time for One Lloyd Iter ($k = n/128$)



[1] Sivic & Zisserman, *Video Google: A text retrieval approach to object matching in videos*
[2] Jégou et al., *Product Quantization for Nearest Neighbor Search*
[3] Chen et al., *SPANN: Highly-Efficient Billion-scale Approximate Nearest Neighbor Search*
[4] Sun et al., *SOAR: Improved Indexing for Approximate Nearest Neighbor Search*

Bottleneck Testing

Typical approach:

Bottleneck Testing

Typical approach:

- **Init centroids with one of:**
  - `k-means++`
  - `k-means||`
  - **uniform random sampling**

Bottleneck Testing

Typical approach:

- Init centroids with one of:
  - `k-means++`
  - `k-means||`
  - uniform random sampling
- **Lloyd's algorithm for local search**

Bottleneck Testing

Typical approach:

- Init centroids with one of:
  - `k-means++`
  - `k-means||`
  - uniform random sampling
- Lloyd's algorithm for local search
- **Iterate until timeout**

Bottleneck Testing

Typical approach:

- Init centroids with one of:
  - `k-means++`
  - `k-means||`
  - uniform random sampling
- Lloyd's algorithm for local search
- Iterate until timeout
- **Accelerate each step on GPU**

Bottleneck Testing

Typical approach:

- Init centroids with one of:
  - `k-means++`
  - `k-means||`
  - uniform random sampling
- Lloyd's algorithm for local search
- Iterate until timeout
- Accelerate each step on GPU

Small k: `k-means++` and `||` are very good

Bottleneck Testing

Typical approach:

- Init centroids with one of:
  - `k-means++`
  - `k-means||`
  - uniform random sampling
- Lloyd's algorithm for local search
- Iterate until timeout
- Accelerate each step on GPU

Small k: `k-means++` and `||` are very good

We have large n and k, e.g. $n = 5e6$ and $k = 1e4$.

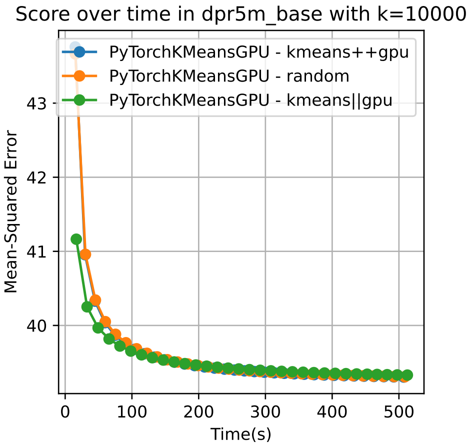
Bottleneck Testing

Typical approach:

- Init centroids with one of:
  - `k-means++`
  - `k-means||`
  - uniform random sampling
- Lloyd's algorithm for local search
- Iterate until timeout
- Accelerate each step on GPU

Small k: `k-means++` and `||` are very good

We have large n and k, e.g. $n = 5e6$ and $k = 1e4$.
Large n/k: Only Lloyd's algorithm matters (right)
Conclusion: Want to accelerate Lloyd's algorithm



Checkpoint 1

Out-of-Core Similarity Search

(used by your favourite semantic search engine)

Basically out-of-core ANNS with n base points

↓

k-Means Clustering

↓

Exact NNS

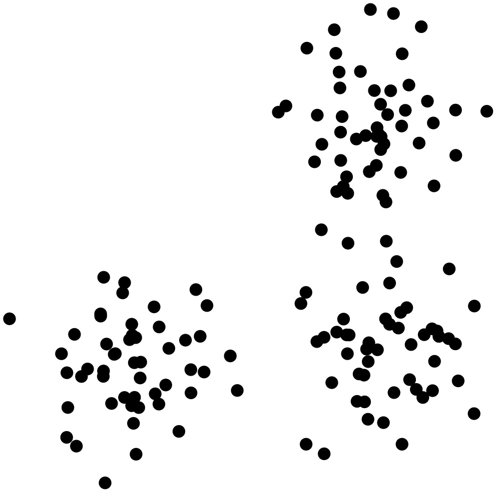
k base points

↓

GPU Acceleration

1. **Initialization:** Sample k centroids uniformly from the dataset.
2. Iterate (local search):
   - **Assignment:** Assign each point to the nearest centroid.
   - **Mean Computation:** Update each centroid to be the average of points assigned to it.

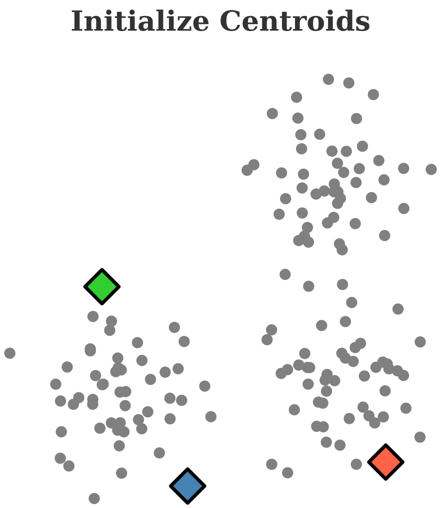
Bottleneck is **Assignment** step.



Lloyd's k-Means Method

1. **Initialization: Sample k centroids uniformly from the dataset.**
2. Iterate (local search):
   - **Assignment:** Assign each point to the nearest centroid.
   - **Mean Computation:** Update each centroid to be the average of points assigned to it.

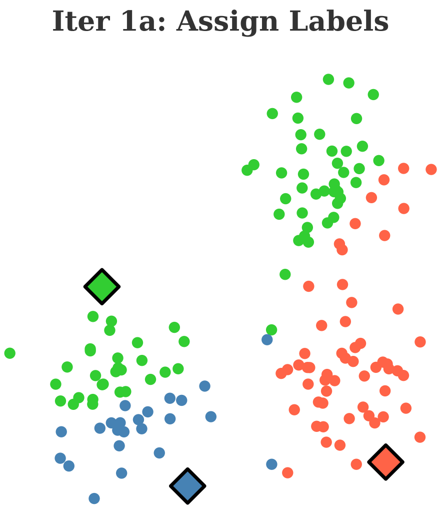
Bottleneck is **Assignment** step.



Lloyd's k-Means Method

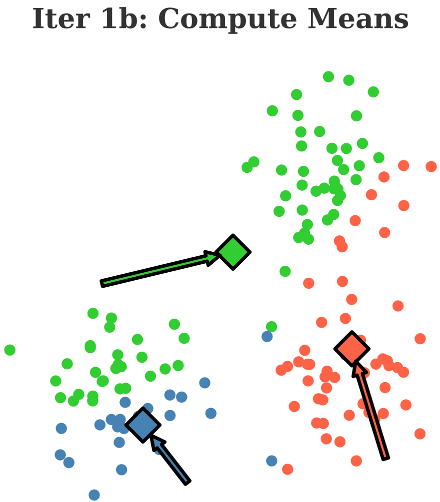
1. **Initialization:** Sample k centroids uniformly from the dataset.
2. Iterate (local search):
   - **Assignment: Assign each point to the nearest centroid.**
   - **Mean Computation:** Update each centroid to be the average of points assigned to it.

Bottleneck is **Assignment** step.



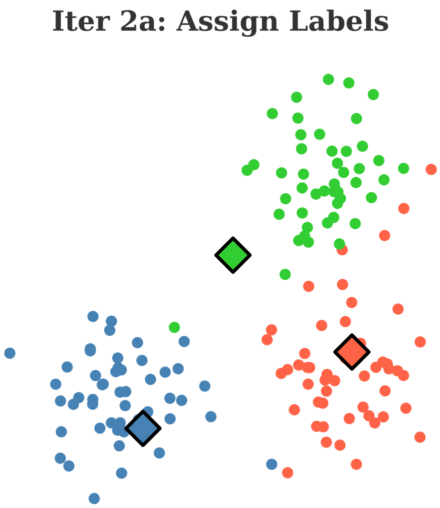
1. **Initialization:** Sample k centroids uniformly from the dataset.
2. Iterate (local search):
   - **Assignment:** Assign each point to the nearest centroid.
   - **Mean Computation: Update each centroid to be the average of points assigned to it.**

Bottleneck is **Assignment** step.



1. **Initialization:** Sample k centroids uniformly from the dataset.
2. Iterate (local search):
   - **Assignment: Assign each point to the nearest centroid.**
   - **Mean Computation:** Update each centroid to be the average of points assigned to it.

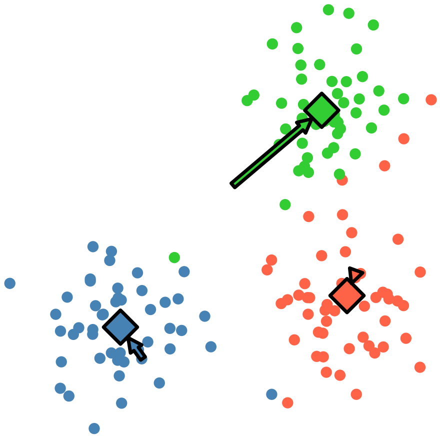
Bottleneck is **Assignment** step.



1. **Initialization:** Sample k centroids uniformly from the dataset.
2. Iterate (local search):
   - **Assignment:** Assign each point to the nearest centroid.
   - **Mean Computation: Update each centroid to be the average of points assigned to it.**

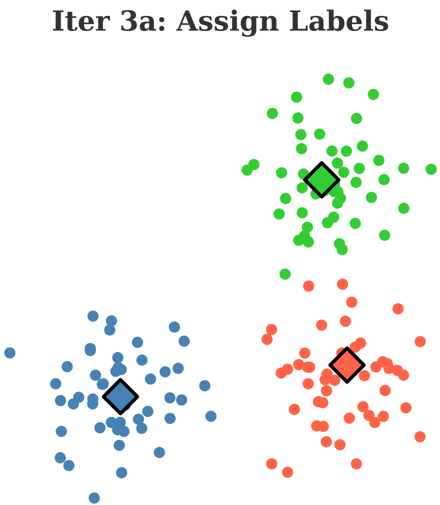
Bottleneck is **Assignment** step.

Iter 2b: Compute Means



1. **Initialization:** Sample k centroids uniformly from the dataset.
2. Iterate (local search):
   - **Assignment: Assign each point to the nearest centroid.**
   - **Mean Computation:** Update each centroid to be the average of points assigned to it.

Bottleneck is **Assignment** step.

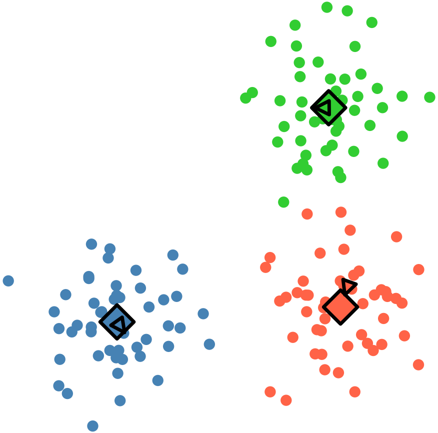


Lloyd's k-Means Method

1. **Initialization:** Sample k centroids uniformly from the dataset.
2. Iterate (local search):
   - **Assignment:** Assign each point to the nearest centroid.
   - **Mean Computation: Update each centroid to be the average of points assigned to it.**

Bottleneck is **Assignment** step.

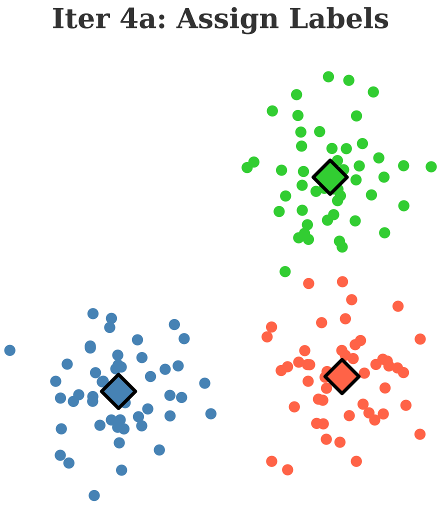
Iter 3b: Compute Means



Lloyd's k-Means Method

1. **Initialization:** Sample k centroids uniformly from the dataset.
2. Iterate (local search):
   - **Assignment: Assign each point to the nearest centroid.**
   - **Mean Computation:** Update each centroid to be the average of points assigned to it.

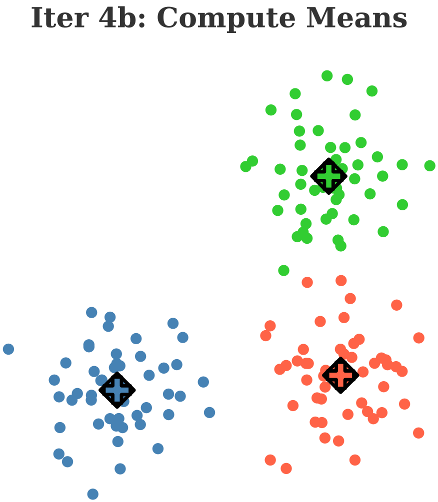
Bottleneck is **Assignment** step.



1. **Initialization:** Sample k centroids uniformly from the dataset.
2. Iterate (local search):
   - **Assignment:** Assign each point to the nearest centroid.
   - **Mean Computation:** Update each centroid to be the average of points assigned to it.

Bottleneck is **Assignment** step.

Key observation: Assignment step is a nearest-neighbour problem.



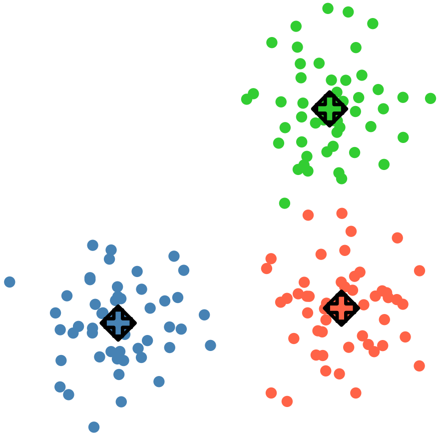
1. **Initialization:** Sample k centroids uniformly from the dataset.
2. Iterate (local search):
   - **Assignment:** Assign each point to the nearest centroid.
   - **Mean Computation:** Update each centroid to be the average of points assigned to it.

Bottleneck is **Assignment** step.

Key observation: Assignment step is a nearest-neighbour problem.

Lloyd's algorithm is very limited in theory.
But it's very good in practice.

Iter 4b: Compute Means



Lloyd Iterations with a Black-box ANNS Data Structure

Alternative iteration approach:

- **Build**: Construct an **approximate nearest neighbor search (ANNS) data structure** on the centroids.
- **Assignment**: **Use the data structure** to assign each data point to its nearest centroid approximately.
- **Mean Computation**: Unchanged

[1] Borodin et al., *Lower Bounds for High Dimensional Nearest Neighbor Search*
[2] Liu, *A strong lower bound for approximate nearest neighbor searching*
[3] Malkov & Yashunin, *Efficient and Robust [ANNS] Using Hierarchical Navigable Small World Graphs*
[4] Raschka et al., *Machine Learning in Python: Main developments and technology trends [...]*

Lloyd Iterations with a Black-box ANNS Data Structure

Alternative iteration approach:

- **Build**: Construct an approximate nearest neighbor search (ANNS) data structure on the centroids.
- **Assignment**: Use the data structure to assign each data point to its nearest centroid approximately.
  - **Make sure not to regress (compare assignments)**
- **Mean Computation**: Unchanged

[1] Borodin et al., *Lower Bounds for High Dimensional Nearest Neighbor Search*

[2] Liu, *A strong lower bound for approximate nearest neighbor searching*

[3] Malkov & Yashunin, *Efficient and Robust [ANNS] Using Hierarchical Navigable Small World Graphs*

[4] Raschka et al., *Machine Learning in Python: Main developments and technology trends [...]*

Lloyd Iterations with a Black-box ANNS Data Structure

Alternative iteration approach:

- **Build**: Construct an approximate nearest neighbor search (ANNS) data structure on the centroids.
- **Assignment**: Use the data structure to assign each data point to its nearest centroid approximately.
  - Make sure not to regress (compare assignments)
- **Mean Computation**: Unchanged

Why ANNS? Exact-NN has a linear lower bound in high-dimensions[1].
ANNS has strong lower bounds too[2], but good widely-used heuristics exist.

---

[1] Borodin et al., *Lower Bounds for High Dimensional Nearest Neighbor Search*
[2] Liu, *A strong lower bound for approximate nearest neighbor searching*
[3] Malkov & Yashunin, *Efficient and Robust [ANNS] Using Hierarchical Navigable Small World Graphs*
[4] Raschka et al., *Machine Learning in Python: Main developments and technology trends [...]*

Lloyd Iterations with a Black-box ANNS Data Structure

Alternative iteration approach:

- **Build**: Construct an approximate nearest neighbor search (ANNS) data structure on the centroids.
- **Assignment**: Use the data structure to assign each data point to its nearest centroid approximately.
  - Make sure not to regress (compare assignments)
- **Mean Computation**: Unchanged

Experiment:

- Baseline: Popular Lloyd implementations:
  - CPU: scikit
  - GPU: cuML, simple pytorch impl
- Suite of (CPU) ANNS data structures:
  - PQ, SQ, IVF IVFPQ, HNSW
  - FAISS implementations

---

[1] Borodin et al., *Lower Bounds for High Dimensional Nearest Neighbor Search*
[2] Liu, *A strong lower bound for approximate nearest neighbor searching*
[3] Malkov & Yashunin, *Efficient and Robust [ANNS] Using Hierarchical Navigable Small World Graphs*
[4] Raschka et al., *Machine Learning in Python: Main developments and technology trends [...]*

Lloyd Iterations with a Black-box ANNS Data Structure

Alternative iteration approach:

- **Build**: Construct an approximate nearest neighbor search (ANNS) data structure on the centroids.
- **Assignment**: Use the data structure to assign each data point to its nearest centroid approximately.
  - Make sure not to regress (compare assignments)
- **Mean Computation**: Unchanged

Experiment:

- **Baseline: Popular Lloyd implementations:**
  - CPU: scikit
  - GPU: cuML, simple pytorch impl
- Suite of (CPU) ANNS data structures:
  - PQ, SQ, IVF IVFPQ, HNSW
  - FAISS implementations

[1] Borodin et al., *Lower Bounds for High Dimensional Nearest Neighbor Search*
[2] Liu, *A strong lower bound for approximate nearest neighbor searching*
[3] Malkov & Yashunin, *Efficient and Robust [ANNS] Using Hierarchical Navigable Small World Graphs*
[4] Raschka et al., *Machine Learning in Python: Main developments and technology trends [...]*

Lloyd Iterations with a Black-box ANNS Data Structure

Alternative iteration approach:

- **Build**: Construct an approximate nearest neighbor search (ANNS) data structure on the centroids.
- **Assignment**: Use the data structure to assign each data point to its nearest centroid approximately.
  - Make sure not to regress (compare assignments)
- **Mean Computation**: Unchanged

Experiment:

- Baseline: Popular Lloyd implementations:
  - CPU: scikit
  - GPU: cuML, simple pytorch impl
- **Suite of (CPU) ANNS data structures:**
  - PQ, SQ, IVF IVFPQ, HNSW
  - FAISS implementations

[1] Borodin et al., *Lower Bounds for High Dimensional Nearest Neighbor Search*
[2] Liu, *A strong lower bound for approximate nearest neighbor searching*
[3] Malkov & Yashunin, *Efficient and Robust [ANNS] Using Hierarchical Navigable Small World Graphs*
[4] Raschka et al., *Machine Learning in Python: Main developments and technology trends [...]*

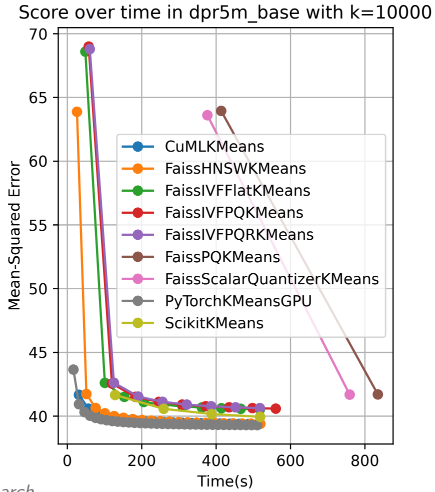
Lloyd Iterations with a Black-box ANNS Data Structure

Alternative iteration approach:

- **Build**: Construct an approximate nearest neighbor search (ANNS) data structure on the centroids.
- **Assignment**: Use the data structure to assign each data point to its nearest centroid approximately.
  - Make sure not to regress (compare assignments)
- **Mean Computation**: Unchanged

Experiment:

- Baseline: Popular Lloyd implementations:
  - CPU: scikit
  - GPU: cuML, simple pytorch impl
- Suite of (CPU) ANNS data structures:
  - PQ, SQ, IVF IVFPQ, HNSW
  - FAISS implementations



[1] Borodin et al., *Lower Bounds for High Dimensional Nearest Neighbor Search*
[2] Liu, *A strong lower bound for approximate nearest neighbor searching*
[3] Malkov & Yashunin, *Efficient and Robust [ANNS] Using Hierarchical Navigable Small World Graphs*
[4] Raschka et al., *Machine Learning in Python: Main developments and technology trends [...]*

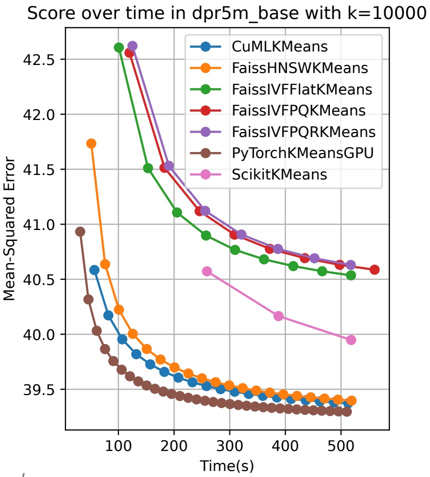
Lloyd Iterations with a Black-box ANNS Data Structure

Alternative iteration approach:

- **Build**: Construct an approximate nearest neighbor search (ANNS) data structure on the centroids.
- **Assignment**: Use the data structure to assign each data point to its nearest centroid approximately.
  - Make sure not to regress (compare assignments)
- **Mean Computation**: Unchanged

Experiment:

- Baseline: Popular Lloyd implementations:
  - CPU: scikit
  - GPU: cuML, simple pytorch impl
- Suite of (CPU) ANNS data structures:
  - PQ, SQ, IVF IVFPQ, HNSW
  - FAISS implementations



[1] Borodin et al., *Lower Bounds for High Dimensional Nearest Neighbor Search*
[2] Liu, *A strong lower bound for approximate nearest neighbor searching*
[3] Malkov & Yashunin, *Efficient and Robust [ANNS] Using Hierarchical Navigable Small World Graphs*
[4] Raschka et al., *Machine Learning in Python: Main developments and technology trends [...]*

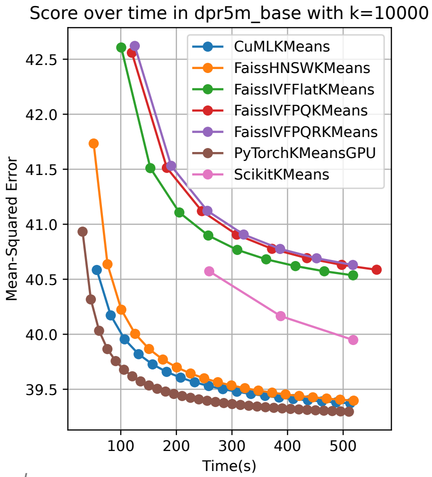
Lloyd Iterations with a Black-box ANNS Data Structure

Alternative iteration approach:

- **Build**: Construct an approximate nearest neighbor search (ANNS) data structure on the centroids.
- **Assignment**: Use the data structure to assign each data point to its nearest centroid approximately.
  - Make sure not to regress (compare assignments)
- **Mean Computation**: Unchanged

Conclusions:

- Dimension reduction (quantization) generally bad



[1] Borodin et al., *Lower Bounds for High Dimensional Nearest Neighbor Search*
[2] Liu, *A strong lower bound for approximate nearest neighbor searching*
[3] Malkov & Yashunin, *Efficient and Robust [ANNS] Using Hierarchical Navigable Small World Graphs*
[4] Raschka et al., *Machine Learning in Python: Main developments and technology trends [...]*

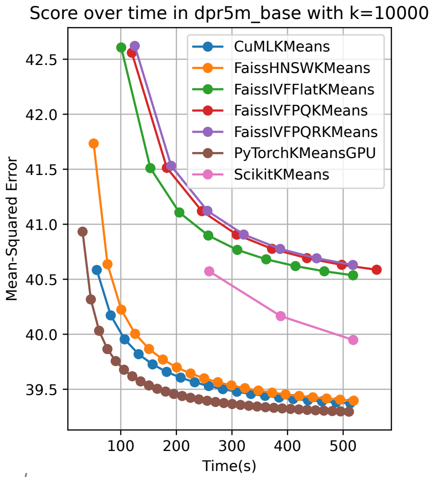
Lloyd Iterations with a Black-box ANNS Data Structure

Alternative iteration approach:

- **Build**: Construct an approximate nearest neighbor search (ANNS) data structure on the centroids.
- **Assignment**: Use the data structure to assign each data point to its nearest centroid approximately.
  - Make sure not to regress (compare assignments)
- **Mean Computation**: Unchanged

Conclusions:

- Dimension reduction (quantization) generally bad
- **HNSW[3] is really good!**



[1] Borodin et al., *Lower Bounds for High Dimensional Nearest Neighbor Search*
[2] Liu, *A strong lower bound for approximate nearest neighbor searching*
[3] Malkov & Yashunin, *Efficient and Robust [ANNS] Using Hierarchical Navigable Small World Graphs*
[4] Raschka et al., *Machine Learning in Python: Main developments and technology trends [...]*

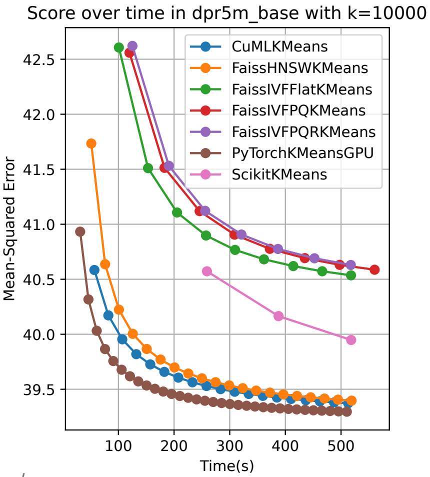
Lloyd Iterations with a Black-box ANNS Data Structure

Alternative iteration approach:

- **Build**: Construct an approximate nearest neighbor search (ANNS) data structure on the centroids.
- **Assignment**: Use the data structure to assign each data point to its nearest centroid approximately.
  - Make sure not to regress (compare assignments)
- **Mean Computation**: Unchanged

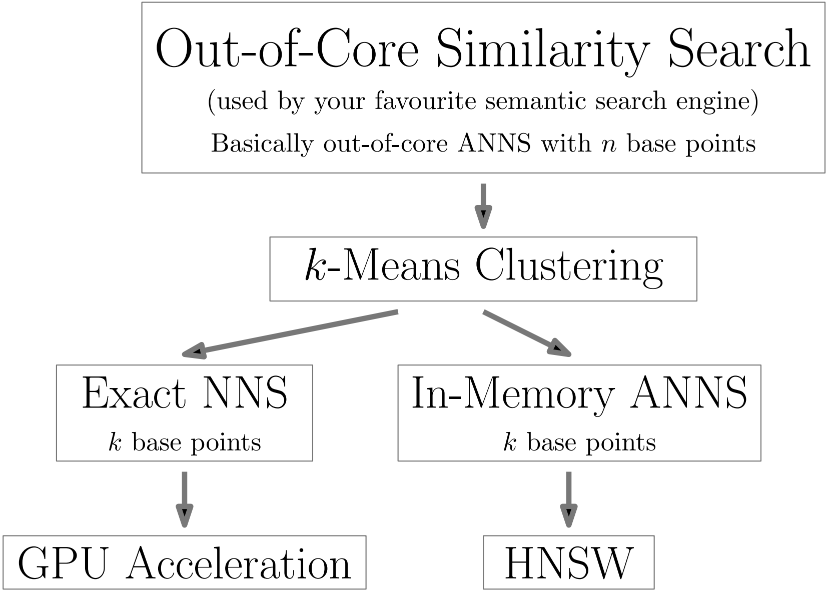
Conclusions:

- Dimension reduction (quantization) generally bad
- HNSW[3] is *really* good!
  - **Almost as good as Nvidia's own GPU imlementation.[4]**

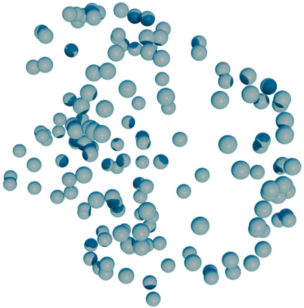


[1] Borodin et al., *Lower Bounds for High Dimensional Nearest Neighbor Search*
[2] Liu, *A strong lower bound for approximate nearest neighbor searching*
[3] Malkov & Yashunin, *Efficient and Robust [ANNS] Using Hierarchical Navigable Small World Graphs*
[4] Raschka et al., *Machine Learning in Python: Main developments and technology trends [...]*

Checkpoint 2



HNSW Review



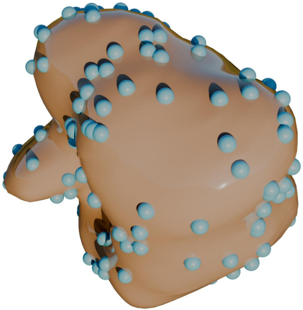
Structure

- Start with (approximate) NN graph
- Prune edges with a heuristic
- Randomly subsample points to get higher layers (similar to skip list)
- Build/insertions also similar to skip list

Search

- Start at arbitrary point on top level
- Greedy local search
- "Seed" lower layers with result

HNSW Review



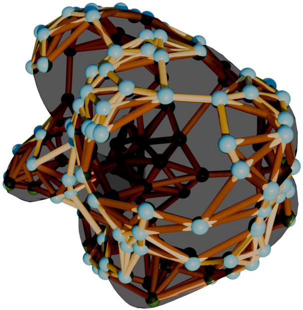
Structure

- Start with (approximate) NN graph
- Prune edges with a heuristic
- Randomly subsample points to get higher layers (similar to skip list)
- Build/insertions also similar to skip list

Search

- Start at arbitrary point on top level
- Greedy local search
- “Seed” lower layers with result

HNSW Review



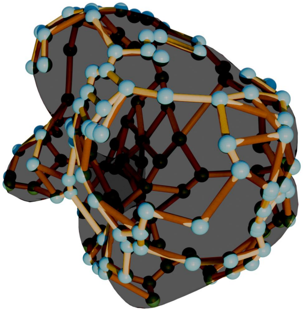
Structure

- **Start with (approximate) NN graph**
- Prune edges with a heuristic
- Randomly subsample points to get higher layers (similar to skip list)
- Build/insertions also similar to skip list

Search

- Start at arbitrary point on top level
- Greedy local search
- "Seed" lower layers with result

HNSW Review



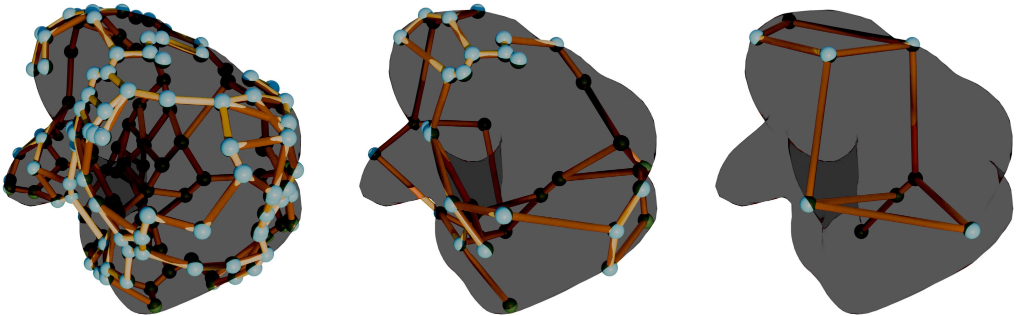
Structure

- Start with (approximate) NN graph
- **Prune edges with a heuristic**
- Randomly subsample points to get higher layers (similar to skip list)
- Build/insertions also similar to skip list

Search

- Start at arbitrary point on top level
- Greedy local search
- "Seed" lower layers with result

HNSW Review



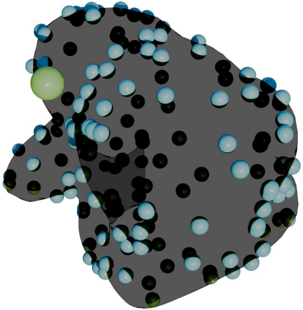
Structure

- Start with (approximate) NN graph
- Prune edges with a heuristic
- **Randomly subsample points to get higher layers (similar to skip list)**
- Build/insertions also similar to skip list

Search

- Start at arbitrary point on top level
- Greedy local search
- "Seed" lower layers with result

HNSW Review



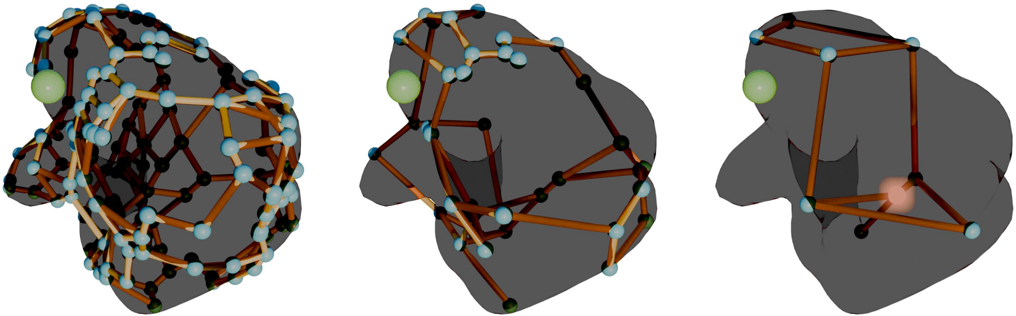
Structure

- Start with (approximate) NN graph
- Prune edges with a heuristic
- Randomly subsample points to get higher layers (similar to skip list)
- Build/insertions also similar to skip list

Search

- Start at arbitrary point on top level
- Greedy local search
- "Seed" lower layers with result

HNSW Review



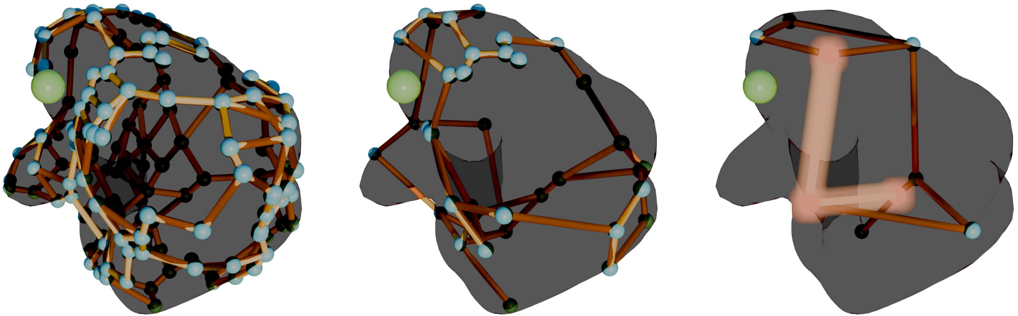
Structure

- Start with (approximate) NN graph
- Prune edges with a heuristic
- Randomly subsample points to get higher layers (similar to skip list)
- Build/insertions also similar to skip list

Search

- **Start at arbitrary point on top level**
- Greedy local search
- "Seed" lower layers with result

HNSW Review



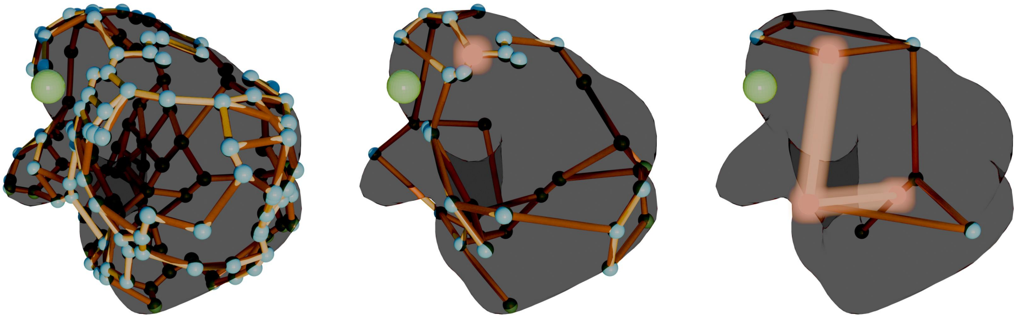
Structure

- Start with (approximate) NN graph
- Prune edges with a heuristic
- Randomly subsample points to get higher layers (similar to skip list)
- Build/insertions also similar to skip list

Search

- Start at arbitrary point on top level
- **Greedy local search**
- "Seed" lower layers with result

HNSW Review



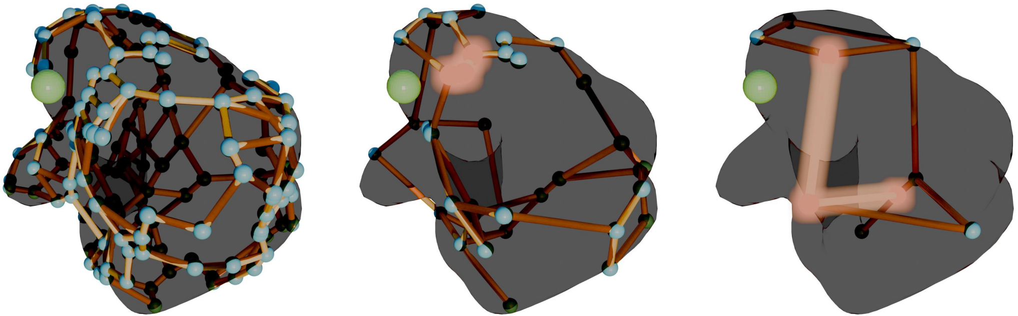
Structure

- Start with (approximate) NN graph
- Prune edges with a heuristic
- Randomly subsample points to get higher layers (similar to skip list)
- Build/insertions also similar to skip list

Search

- Start at arbitrary point on top level
- Greedy local search
- **"Seed" lower layers with result**

HNSW Review



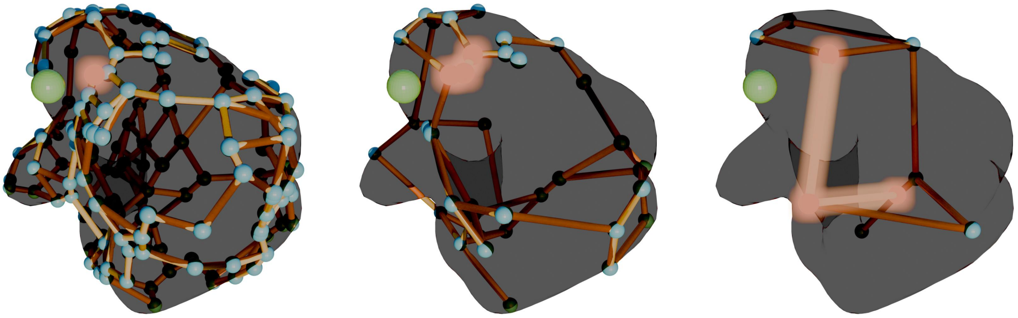
Structure

- Start with (approximate) NN graph
- Prune edges with a heuristic
- Randomly subsample points to get higher layers (similar to skip list)
- Build/insertions also similar to skip list

Search

- Start at arbitrary point on top level
- **Greedy local search**
- "Seed" lower layers with result

HNSW Review



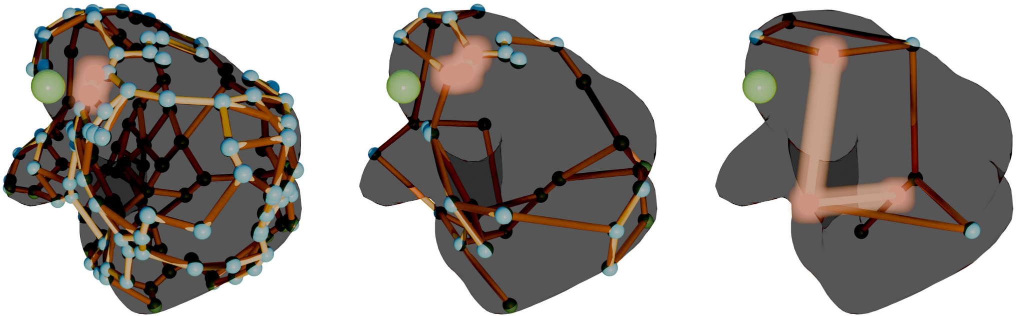
Structure

- Start with (approximate) NN graph
- Prune edges with a heuristic
- Randomly subsample points to get higher layers (similar to skip list)
- Build/insertions also similar to skip list

Search

- Start at arbitrary point on top level
- Greedy local search
- **"Seed" lower layers with result**

HNSW Review



Structure

- Start with (approximate) NN graph
- Prune edges with a heuristic
- Randomly subsample points to get higher layers (similar to skip list)
- Build/insertions also similar to skip list

Search

- Start at arbitrary point on top level
- **Greedy local search**
- "Seed" lower layers with result

Seeded ANNS

Normal ANNS queries: given a query point, find a good close neighbor.

Seeded ANNS

Normal ANNS queries: given a query point, find a good close neighbor.

Seeded ANNS queries: also given **seed points** (candidate close neighbors).
Very simple learning-augmented form of ANNS.

Seeded ANNS

Normal ANNS queries: given a query point, find a good close neighbor.

Seeded ANNS queries: also given **seed points** (candidate close neighbors).
Very simple learning-augmented form of ANNS.
Want:

- robustness — good solutions even if seed points bad
- consistency — better solutions if seed points good

Seeded ANNS

Normal ANNS queries: given a query point, find a good close neighbor.

Seeded ANNS queries: also given **seed points** (candidate close neighbors).
Very simple learning-augmented form of ANNS.
Want:

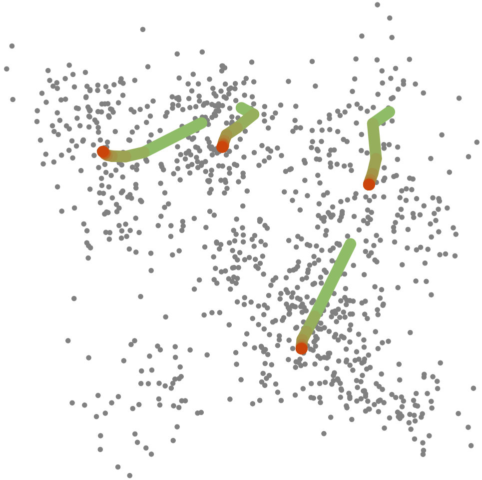
- robustness — good solutions even if seed points bad
- consistency — better solutions if seed points good

In HNSW: Add seed points right at start of search on last level. Similar for other search-graph methods (result: **seeded search-graphs**).

Main Improvements: Beating Black-Box HNSW

Centroids "slow down" over time:

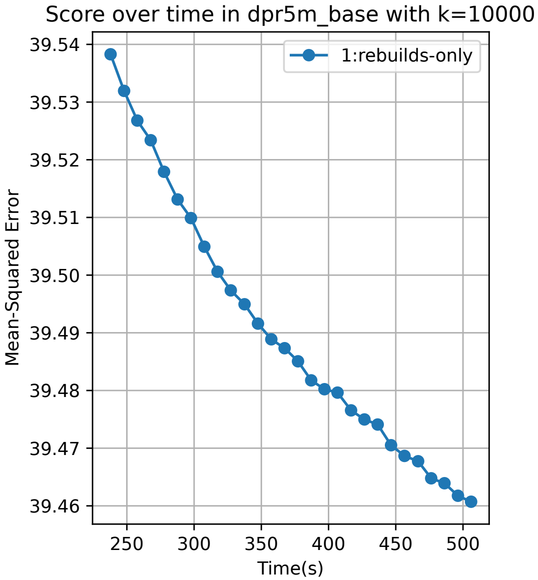
- Rebuilds (HNSW as a kind of kinetic data structure, omitting details)
- Extra "seed points" from prev assignment: Seeded ANNS



Main Improvements: Beating Black-Box HNSW

Centroids "slow down" over time:

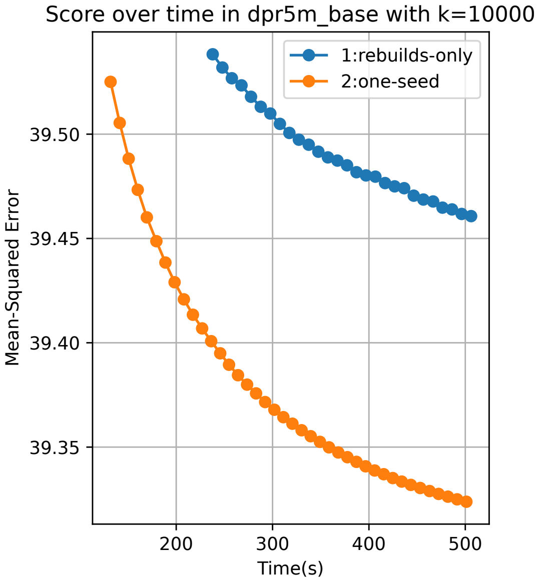
- **Rebuilds** (HNSW as a kind of kinetic data structure, omitting details)
- Extra "seed points" from prev assignment: Seeded ANNS



Main Improvements: Beating Black-Box HNSW

Centroids "slow down" over time:

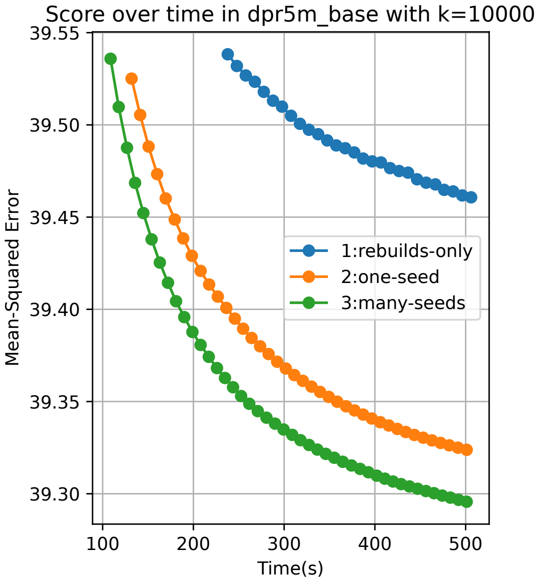
- Rebuilds (HNSW as a kind of kinetic data structure, omitting details)
- **Extra "seed points" from prev assignment: Seeded ANNS**



Main Improvements: Beating Black-Box HNSW

Centroids "slow down" over time:

- Rebuilds (HNSW as a kind of kinetic data structure, omitting details)
- **Multiple** extra "seed points" from prev assignment: Seeded ANNS



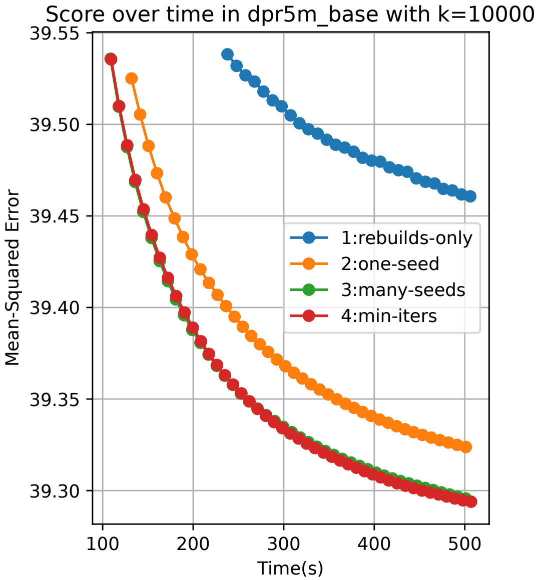
Main Improvements: Beating Black-Box HNSW

Centroids “slow down” over time:

- Rebuilds (HNSW as a kind of kinetic data structure, omitting details)
- Multiple extra “seed points” from prev assignment: Seeded ANNS

Often also improve:

- **Min iteration threshold**
- Bulk queries for more seed points



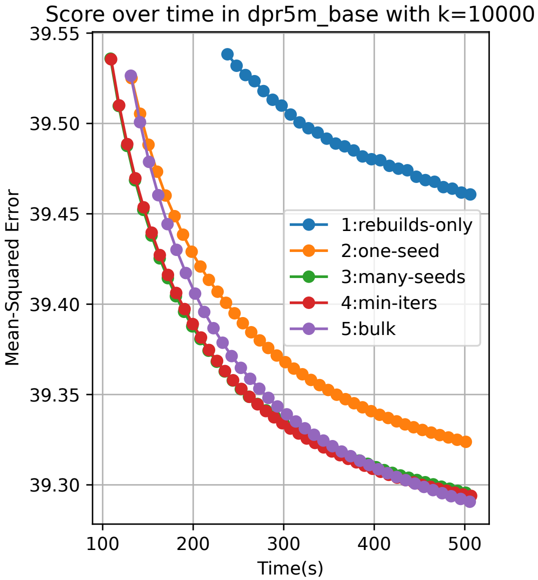
Main Improvements: Beating Black-Box HNSW

Centroids "slow down" over time:

- Rebuilds (HNSW as a kind of kinetic data structure, omitting details)
- Multiple extra "seed points" from prev assignment: Seeded ANNS

Often also improve:

- Min iteration threshold
- **Bulk queries for more seed points**



Main Improvements: Beating Black-Box HNSW

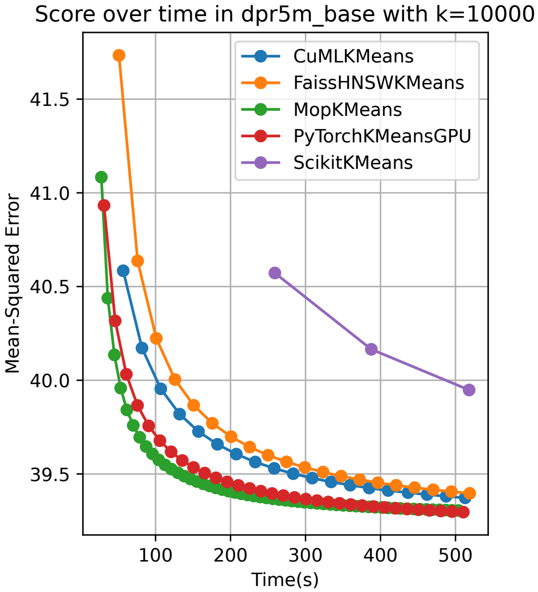
Centroids "slow down" over time:

- Rebuilds (HNSW as a kind of kinetic data structure, omitting details)
- Multiple extra "seed points" from prev assignment: Seeded ANNS

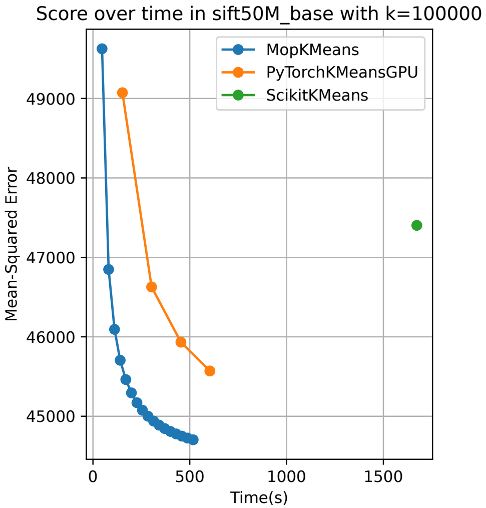
Often also improve:

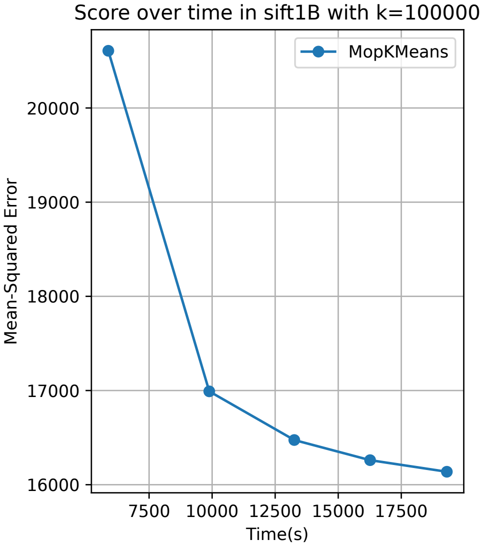
- Min iteration threshold
- Bulk queries for more seed points

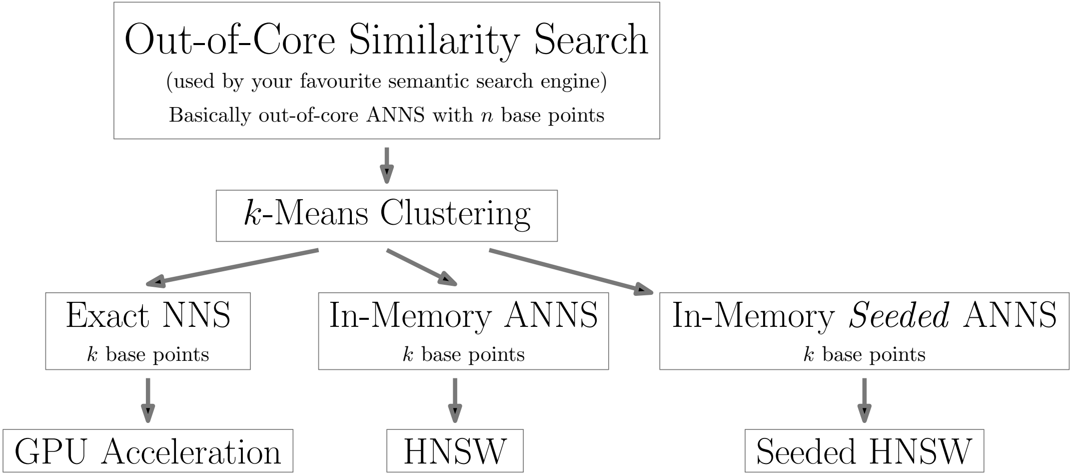
Now (mostly) **beating GPU implementations with CPU**.



More Results



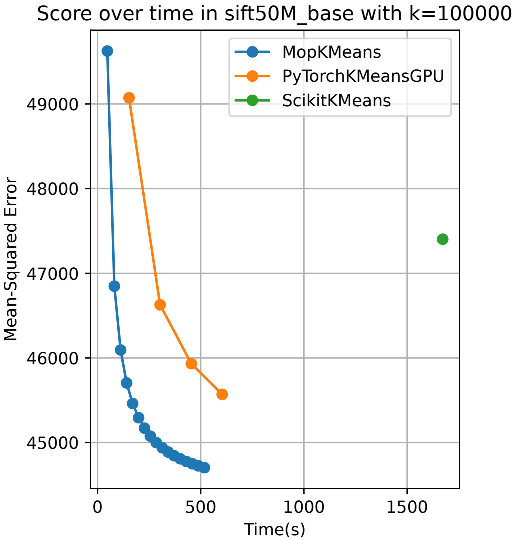



Out-of-Core Similarity Search
(used by your favourite semantic search engine)
Basically out-of-core ANNS with n base points
k-Means Clustering
Exact NNS
k base points
In-Memory ANNS
k base points
In-Memory *Seeded* ANNS
k base points
GPU Acceleration
HNSW
Seeded HNSW

Open Problems

- GPU-acceleration of SANNS? No current methods for ANNS on GPU extend well.
- Perfect consistency guarantees for SANNS in fixed doubling dimension? Search-graph methods don't seem to work.
- General open ANNS problem: Better theoretical understanding of why search-graphs work well? Best known is (tunable) additive approximation known for one specific graph algorithm, with fixed doubling-dimension[1].

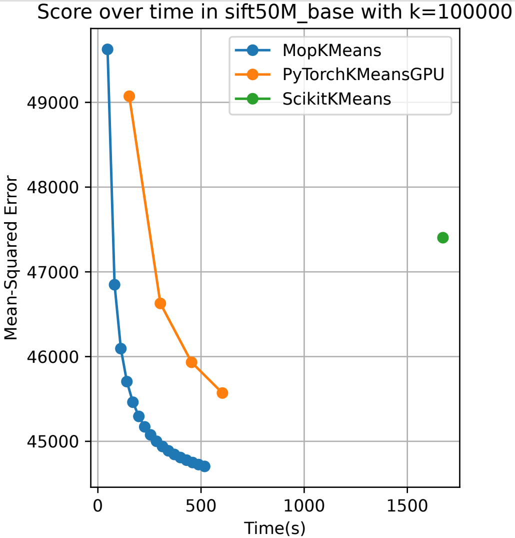
arxiv.org/abs/2502.06163



[1] Indyk & Xu, *Worst-case performance of popular approximate nearest neighbor search implementations [...]*

Open Problems

- GPU-acceleration of SANNS? No current methods for ANNS on GPU extend well.
- Perfect consistency guarantees for SANNS in fixed doubling dimension? Search-graph methods don't seem to work.
- General open ANNS problem: Better theoretical understanding of why search-graphs work well? Best known is (tunable) additive approximation known for one specific graph algorithm, with fixed doubling-dimension[1].



arxiv.org/abs/2502.06163

Fin.

[1] Indyk & Xu, *Worst-case performance of popular approximate nearest neighbor search implementations [...]*