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

**Independent**

# MOPBucket: A Massively OP algorithm for $k$-means clustering Bucketloads of data

**Joint work with Eliot Robson and Da Wei Zheng**

FWCG 2024

# $k$-Means/Sum of Squares Clustering: Quick Review

## Input

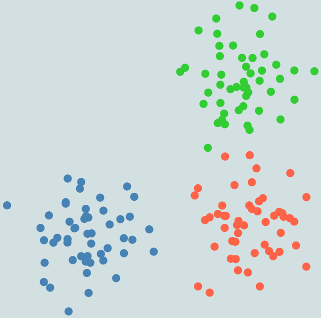
- $k$: a parameter (e.g., $k = 3$)
- $n$ $d$-dimensional vectors

# $k$-Means/Sum of Squares Clustering: Quick Review

**Input**

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

**Output**



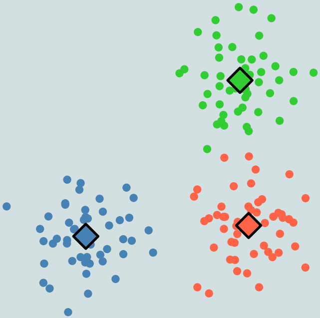
- Clusters $\{C_i\}_{i\in[k]}$ with centroids $\{\mu_i\}_{i\in[k]}$
- Objective: $\min \sum_{i=1}^{k} \sum_{x,y\in C_i} \|x-y\|^2$

# $k$-Means/Sum of Squares Clustering: Quick Review

**Input**

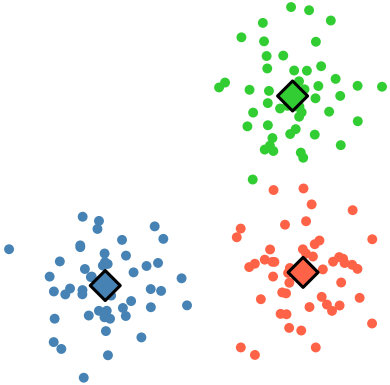
- $k$: a parameter (e.g., $k = 3$)
- $n$ $d$-dimensional vectors

**Output**



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

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

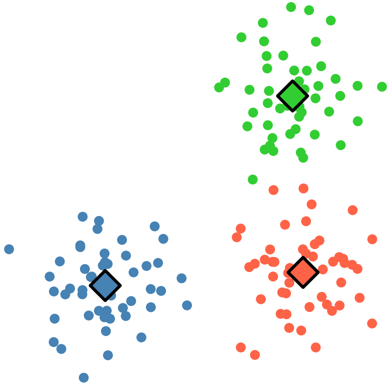


[1] Aloise et al., *NP-hardness of Euclidean sum-of-squares clustering*
[2] Arthur & Vassilvitskii, *k-means++: The Advantages of Careful Seeding*
[3] Bahmani et al., *Scalable k-means++*
[4] Lloyd, *Least square quantization in PCM*
[5] Bahmani's slides on `k-means||`

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

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

**Two broad approaches for good solutions:**



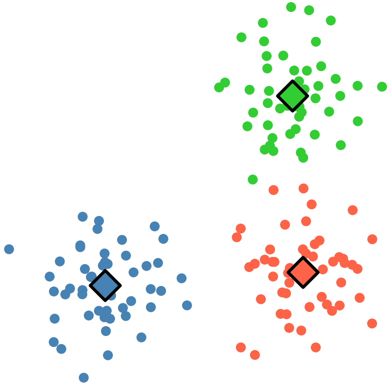
[1] Aloise et al., *NP-hardness of Euclidean sum-of-squares clustering*
[2] Arthur & Vassilvitskii, *k-means++: The Advantages of Careful Seeding*
[3] Bahmani et al., *Scalable k-means++*
[4] Lloyd, *Least square quantization in PCM*
[5] Bahmani's slides on `k-means||`

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

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

Two broad approaches for good solutions:

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



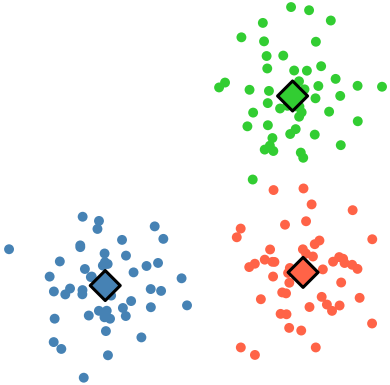
[1] Aloise et al., *NP-hardness of Euclidean sum-of-squares clustering*
[2] Arthur & Vassilvitskii, *k-means++: The Advantages of Careful Seeding*
[3] Bahmani et al., *Scalable k-means++*
[4] Lloyd, *Least square quantization in PCM*
[5] Bahmani's slides on `k-means||`

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

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

Two broad approaches for good solutions:

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



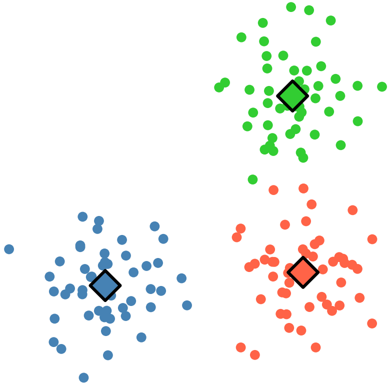
[1] Aloise et al., *NP-hardness of Euclidean sum-of-squares clustering*
[2] Arthur & Vassilvitskii, *k-means++: The Advantages of Careful Seeding*
[3] Bahmani et al., *Scalable k-means++*
[4] Lloyd, *Least square quantization in PCM*
[5] Bahmani's slides on `k-means||`

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

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

Two broad approaches for good 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



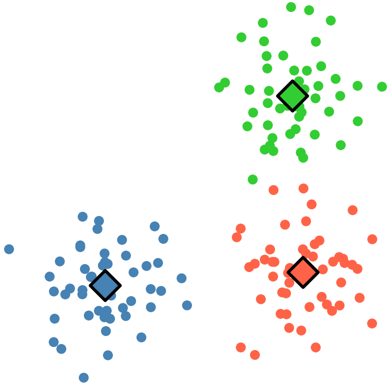
[1] Aloise et al., *NP-hardness of Euclidean sum-of-squares clustering*
[2] Arthur & Vassilvitskii, *k-means++: The Advantages of Careful Seeding*
[3] Bahmani et al., *Scalable k-means++*
[4] Lloyd, *Least square quantization in PCM*
[5] Bahmani's slides on `k-means||`

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

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

Two broad approaches for good 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||`

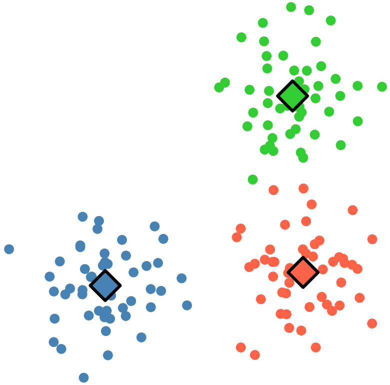
# $k$-Means/Sum of Squares Clustering: Complexity and Approaches

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

Two broad approaches for good 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*

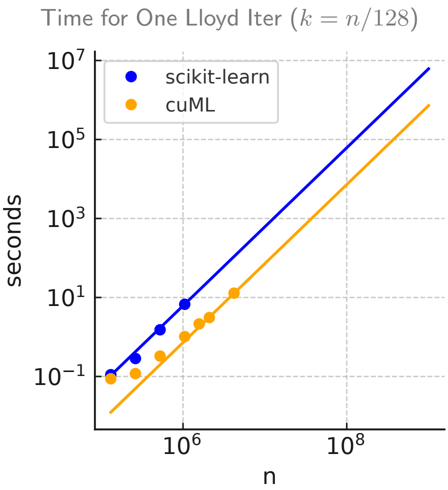
[2] Arthur & Vassilvitskii, *k-means++: The Advantages of Careful Seeding*

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

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

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

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

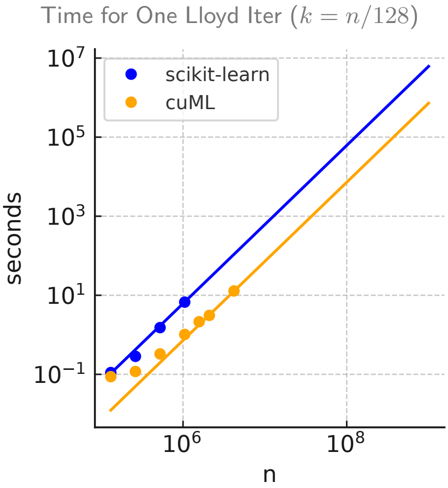



[1] Sivic & Zisserman, *Video Google: A text retrieval approach to object matching in videos*
[2] Jégou et al., *Product Quantization for Nearest Neighbor Search*
[3] Chen et al., *SPANN: Highly-Efficient Billion-scale Approximate Nearest Neighbor Search*
[4] Sun et al., *SOAR: Improved Indexing for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

# Some Motivation

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

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



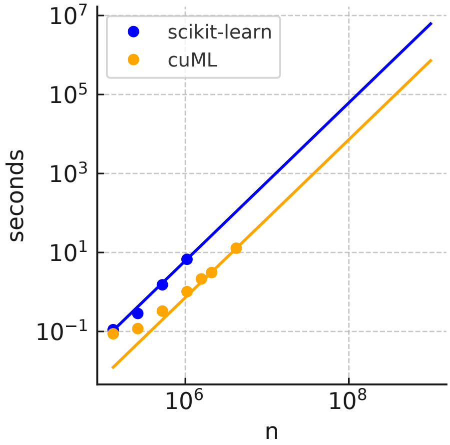

[1] Sivic & Zisserman, *Video Google: A text retrieval approach to object matching in videos*
[2] Jégou et al., *Product Quantization for Nearest Neighbor Search*
[3] Chen et al., *SPANN: Highly-Efficient Billion-scale Approximate Nearest Neighbor Search*
[4] Sun et al., *SOAR: Improved Indexing for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

# 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.**

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

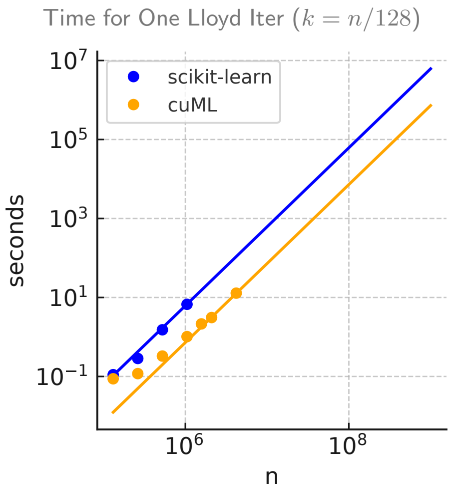



[1]Sivic & Zisserman, *Video Google: A text retrieval approach to object matching in videos*
[2]Jégou et al., *Product Quantization for Nearest Neighbor Search*
[3]Chen et al., *SPANN: Highly-Efficient Billion-scale Approximate Nearest Neighbor Search*
[4]Sun et al., *SOAR: Improved Indexing for Approximate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



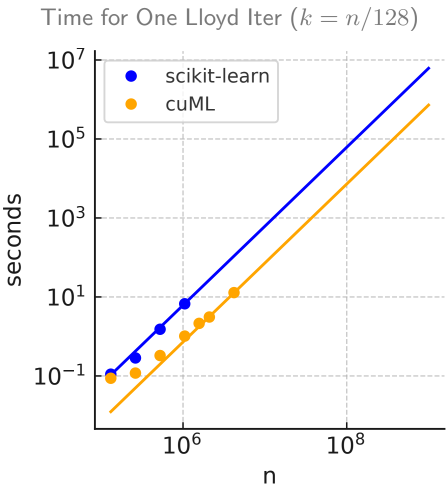

[1] Sivic & Zisserman, *Video Google: A text retrieval approach to object matching in videos*
[2] Jégou et al., *Product Quantization for Nearest Neighbor Search*
[3] Chen et al., *SPANN: Highly-Efficient Billion-scale Approximate Nearest Neighbor Search*
[4] Sun et al., *SOAR: Improved Indexing for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

All methods on the previous slide have a linear dependence on $k$.

**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.

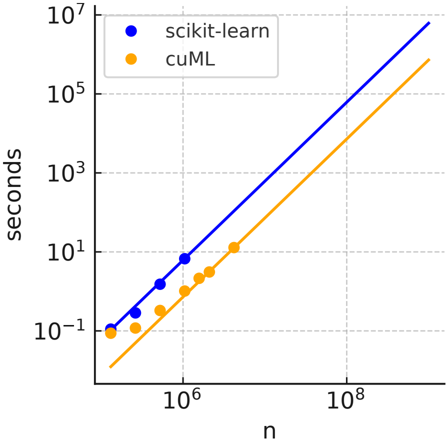
- Many methods used in industry for massive datasets[1,2,3,4] solve $k$-means as a sub-routine, and would benefit from very large $k$.
- Many would benefit from large $k$ ($k \approx n/c$ for small $c$), but current methods are too slow.

All methods on the previous slide have a linear dependence on $k$.

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

- **Would take weeks with currently 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*

# 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.

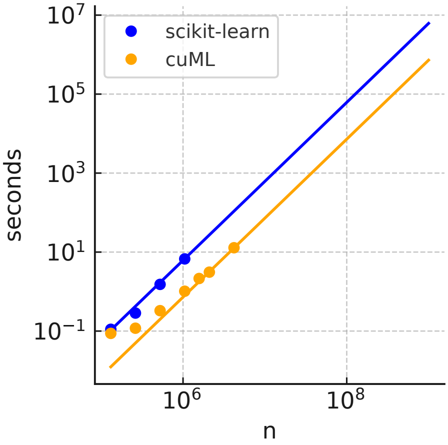
All methods on the previous slide have a linear dependence on $k$.

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

- Would take weeks with currently 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 $k = 1e4$.**

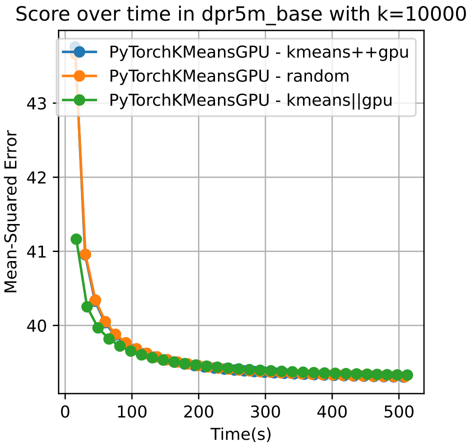
# Bottleneck Testing

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

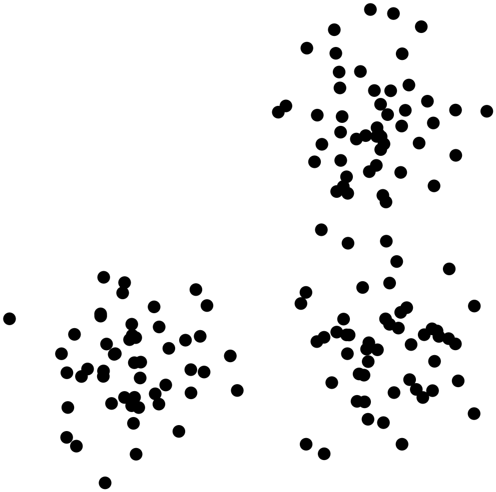
We have large $n$ and $k$, e.g. $n = 5e6$ and $k = 1e4$.
**Large $n/k$: Only Lloyd's algorithm matters (right)**
**Conclusion: Want to accelerate Lloyd's algorithm**



# 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.

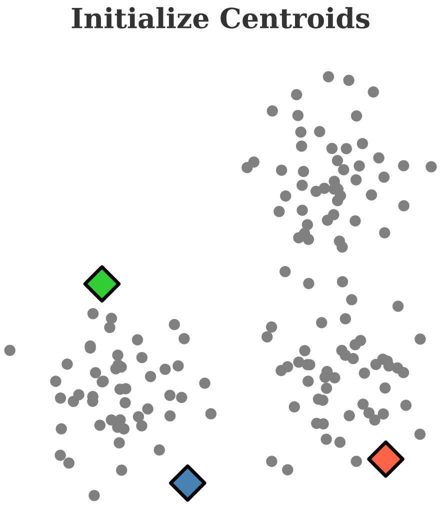
Bottleneck is **Assignment** step.



# Lloyd's $k$-Means Method

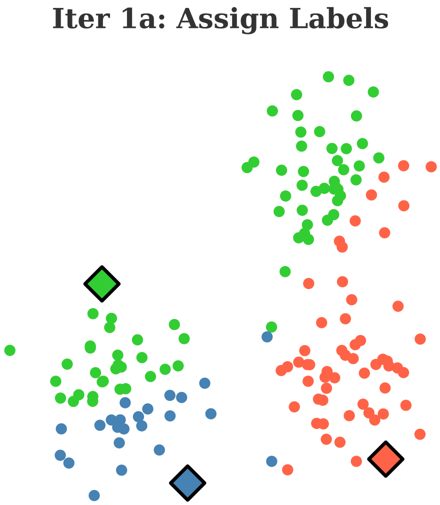
1. **Initialization: Sample $k$ centroids uniformly from the dataset.**
2. Iterate (local search):
   - **Assignment:** Assign each point to the nearest centroid.
   - **Mean Computation:** Update each centroid to be the average of points assigned to it.

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.

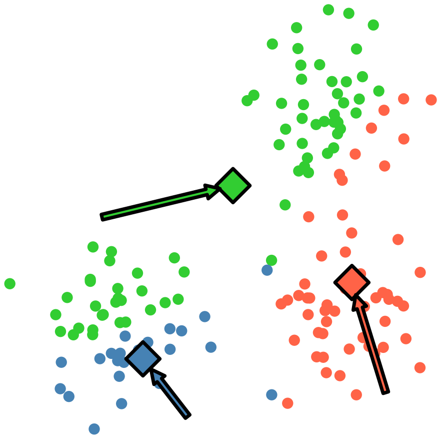
Bottleneck is **Assignment** step.



# 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.**

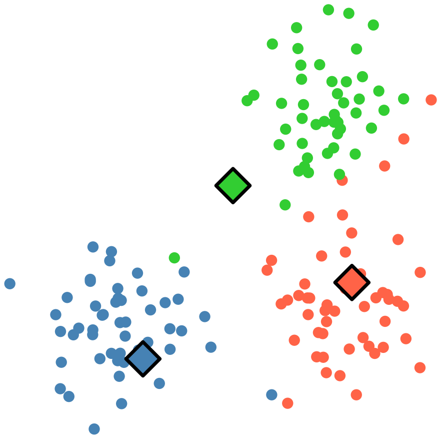
Bottleneck is **Assignment** step.



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.

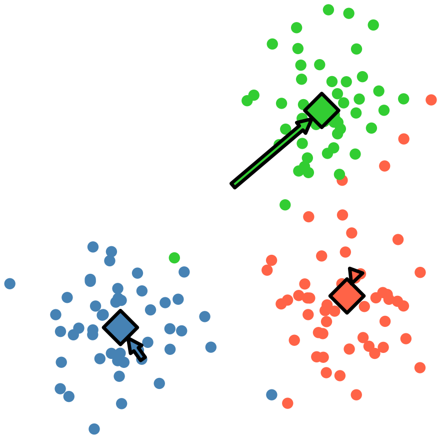
**Iter 2a: Assign Labels**



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.**

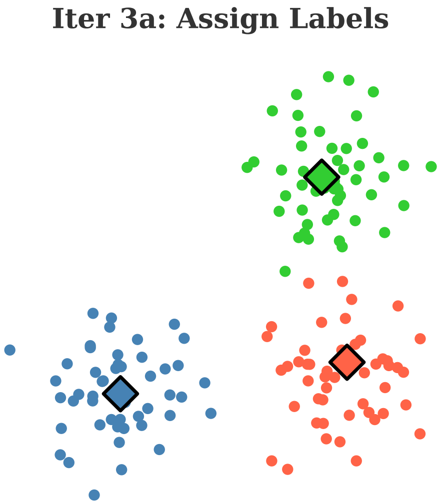
Bottleneck is **Assignment** step.

### 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.

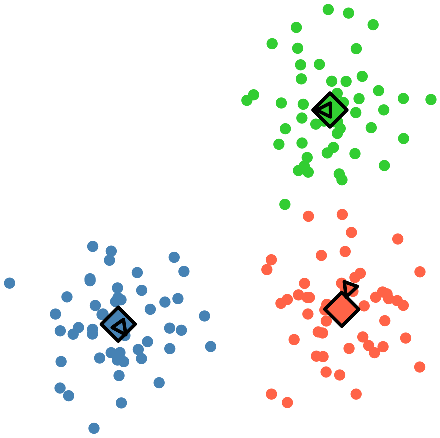


# Lloyd's $k$-Means Method

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.

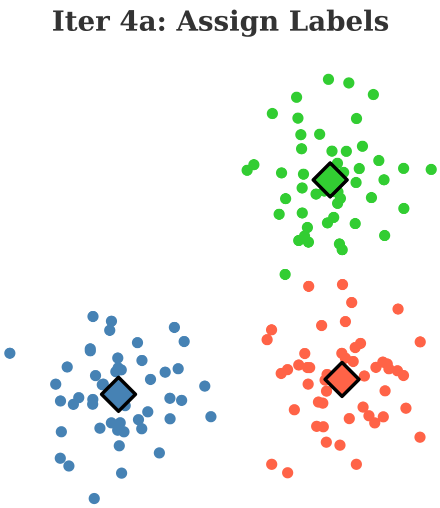
## Iter 3b: Compute Means



# 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.

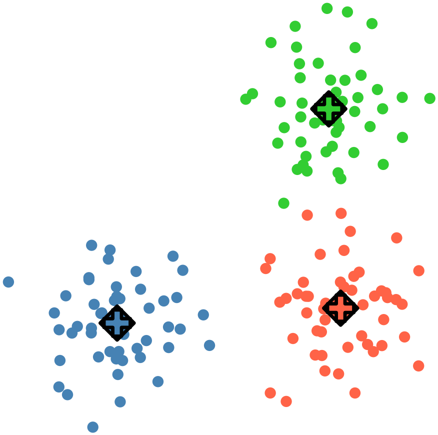


1. **Initialization:** Sample $k$ centroids uniformly from the dataset.
2. Iterate (local search):
   - **Assignment:** Assign each point to the nearest centroid.
   - **Mean Computation:** Update each centroid to be the average of points assigned to it.

Bottleneck is **Assignment** step.

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

**Iter 4b: Compute Means**



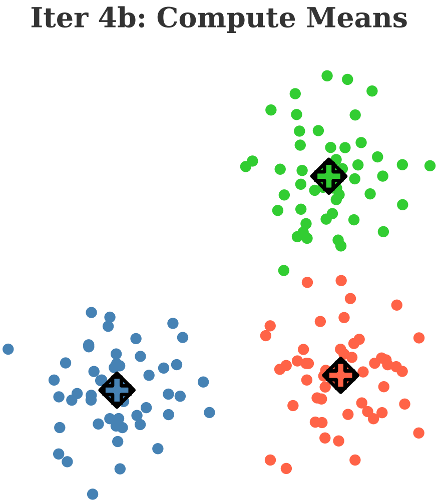
# Lloyd's $k$-Means Method

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.

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

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



# Lloyd Iterations with a Black-box ANNS Data Structure

(Surprisingly) new 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

(Surprisingly) new 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

(Surprisingly) new 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

(Surprisingly) new 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

(Surprisingly) new 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

(Surprisingly) new 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 [...]*

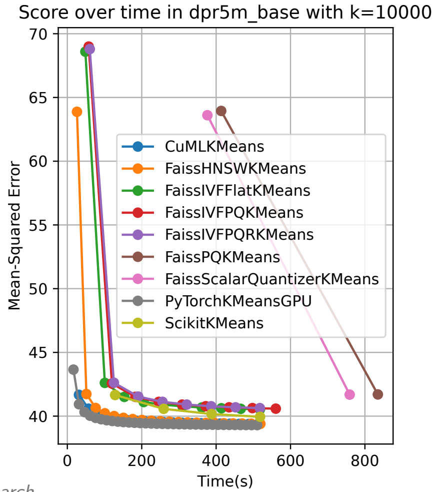
# Lloyd Iterations with a Black-box ANNS Data Structure

(Surprisingly) new 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 [...]*

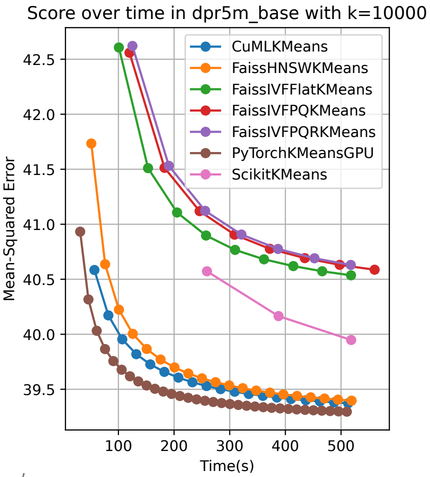
# Lloyd Iterations with a Black-box ANNS Data Structure

(Surprisingly) new 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 [...]*

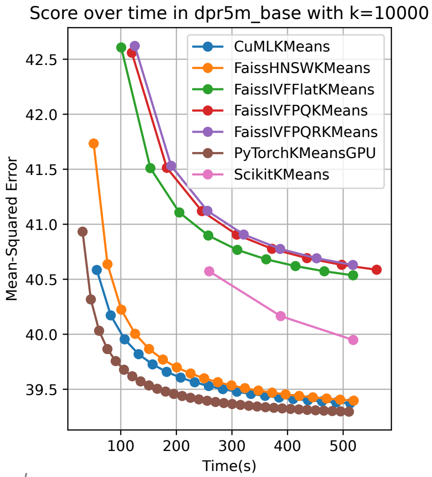
# Lloyd Iterations with a Black-box ANNS Data Structure

(Surprisingly) new 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 [...]*

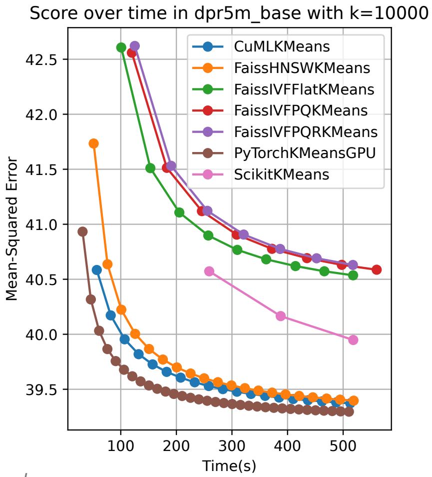
# Lloyd Iterations with a Black-box ANNS Data Structure

(Surprisingly) new 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 [...]*

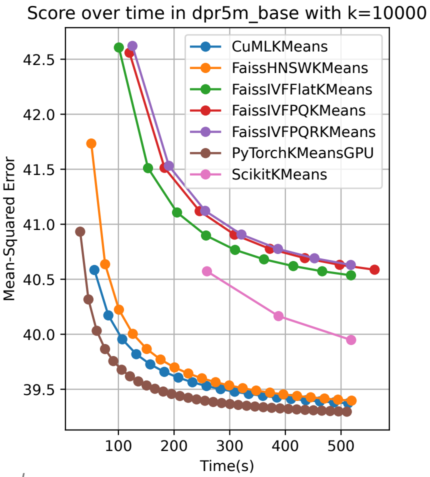
# Lloyd Iterations with a Black-box ANNS Data Structure

(Surprisingly) new 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]**



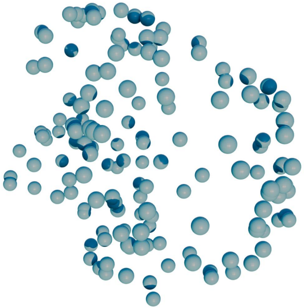

[1] Borodin et al., *Lower Bounds for High Dimensional Nearest Neighbor Search*

[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 [...]*

# HNSW Review



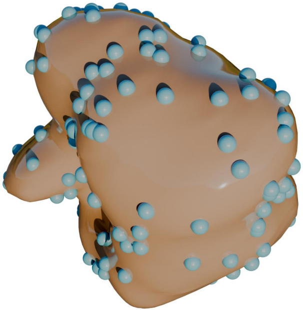
**Structure**

- 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



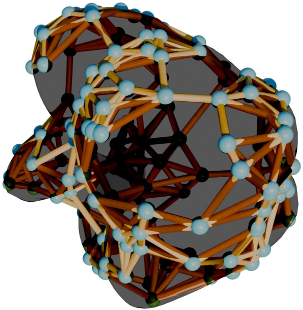
**Structure**

- 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



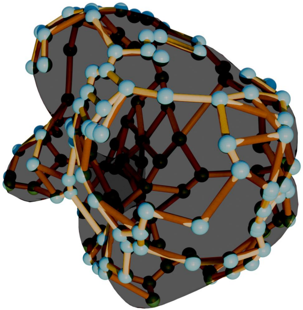
**Structure**

- **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



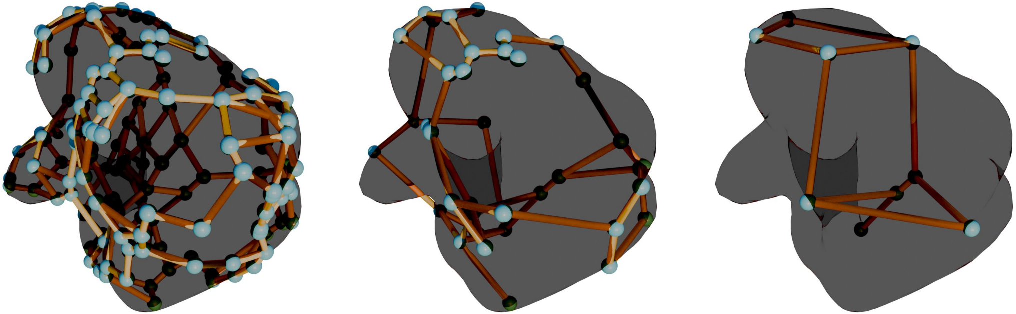
**Structure**

- 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



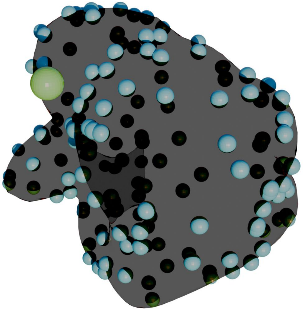
**Structure**

- 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



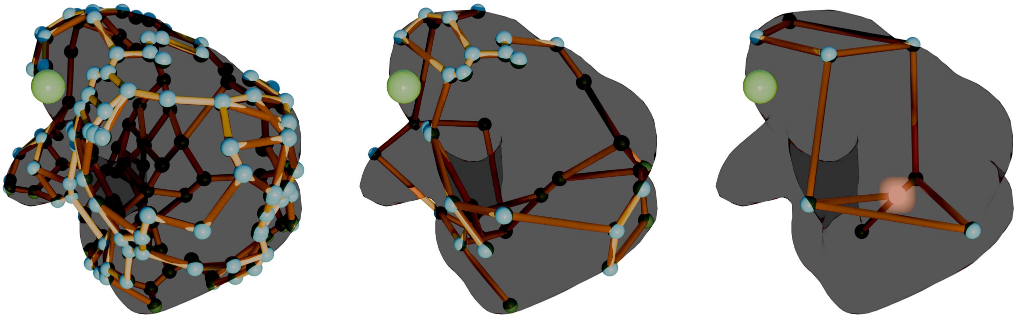
### Structure

- 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



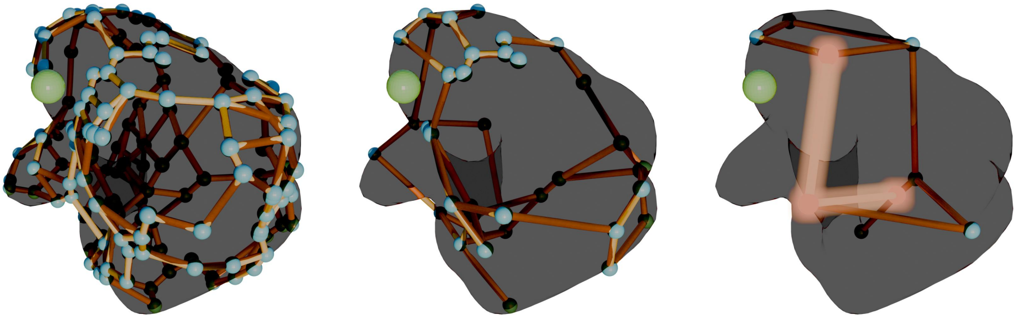
**Structure**

- 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



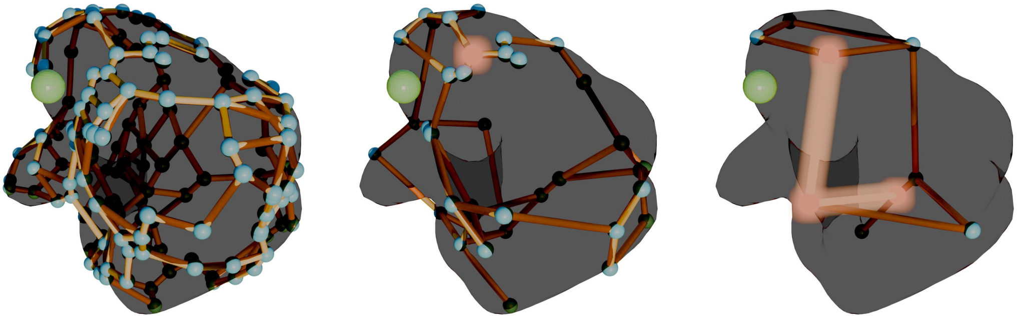
**Structure**

- 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



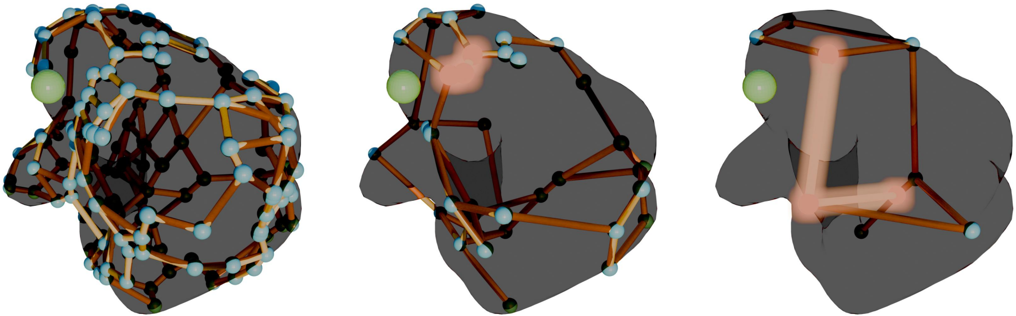
**Structure**

- 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

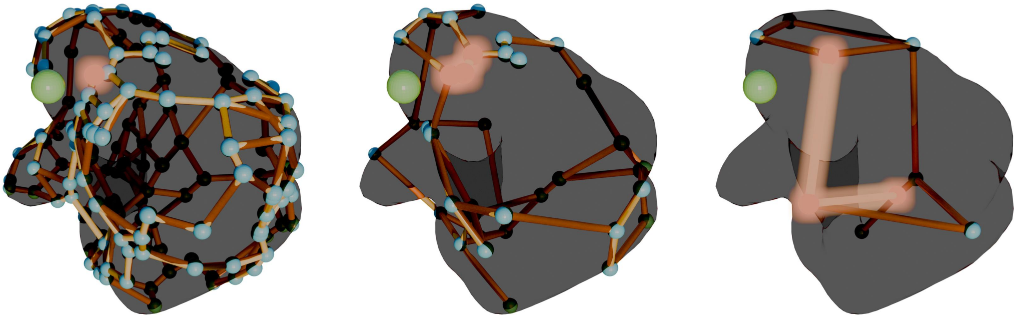


**Structure**

- 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



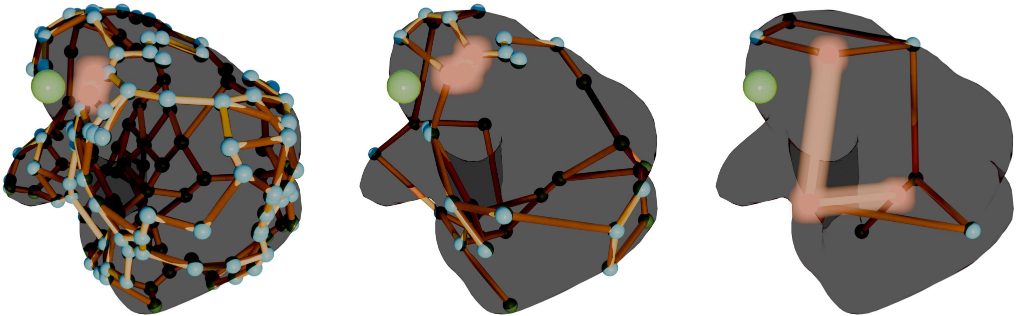
**Structure**

- 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**

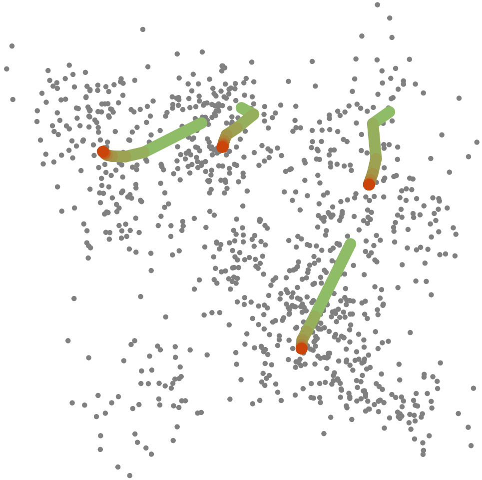
- Start with (approximate) NN graph
- Prune edges with a heuristic
- Randomly subsample points to get higher layers (similar to skip list)
- Build/insertions also similar to skip list

**Search**

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

**Centroids "slow down" over time**:

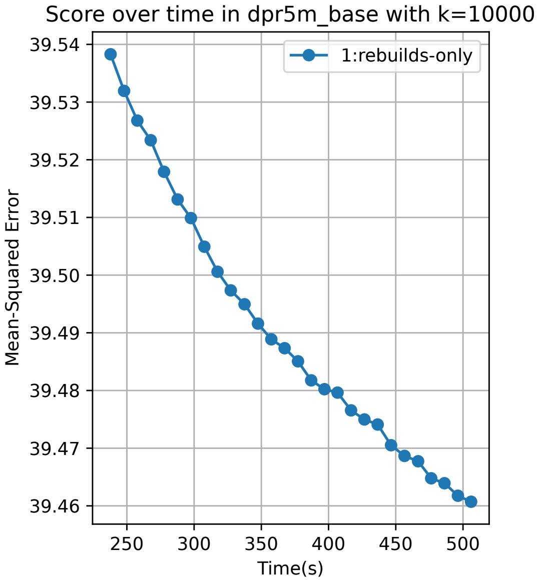
- Rebuilds (omitting details)
- Extra "seed points" from prev assignment



# Main Improvements: Beating Black-Box HNSW

Centroids "slow down" over time:

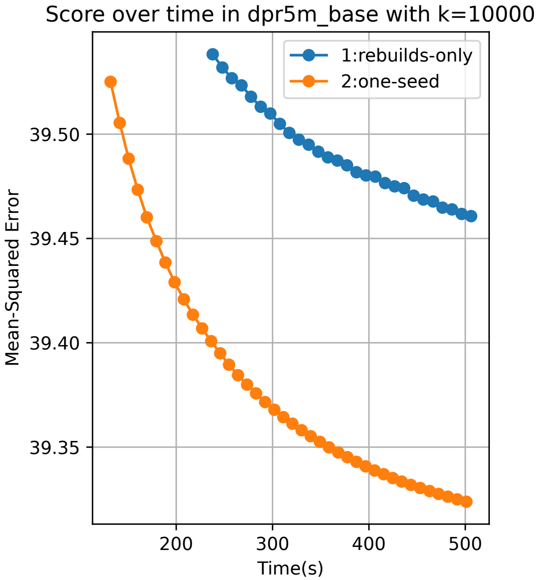
- **Rebuilds** (omitting details)
- Extra "seed points" from prev assignment



# Main Improvements: Beating Black-Box HNSW

Centroids "slow down" over time:

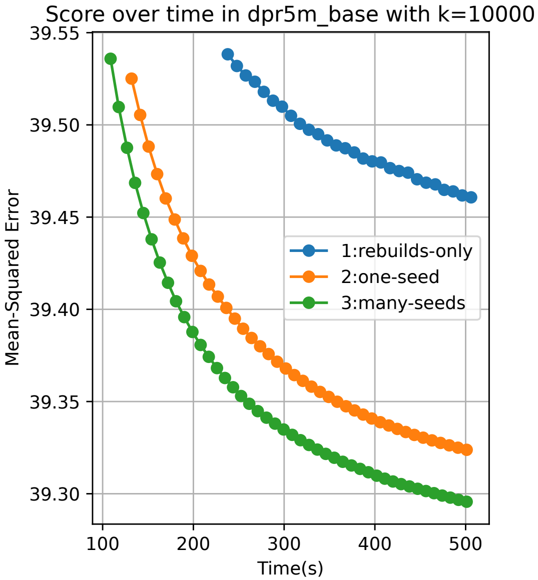
- Rebuilds (omitting details)
- **Extra "seed points" from prev assignment**



# Main Improvements: Beating Black-Box HNSW

Centroids "slow down" over time:

- Rebuilds (omitting details)
- **Multiple** extra "seed points" from prev assignment



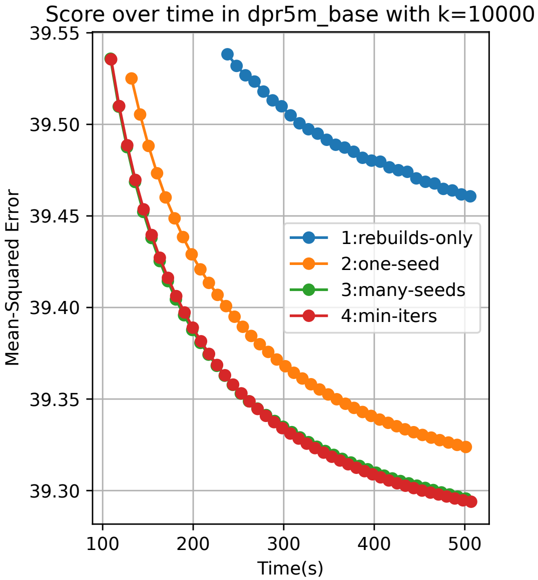
# Main Improvements: Beating Black-Box HNSW

Centroids “slow down” over time:

- Rebuilds (omitting details)
- Multiple extra “seed points” from prev assignment

Often also improve:

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



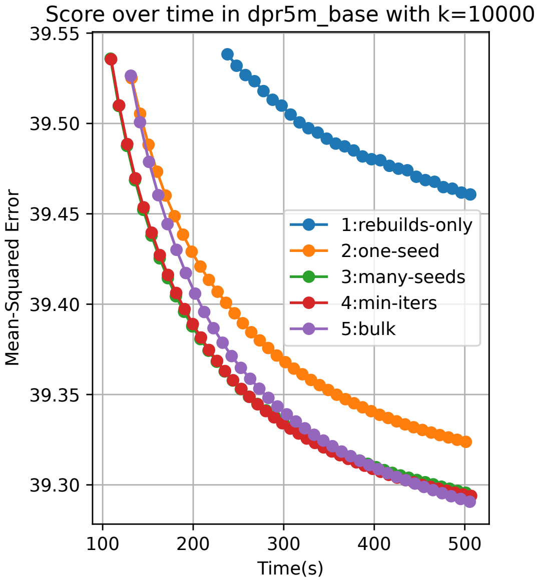
# Main Improvements: Beating Black-Box HNSW

Centroids "slow down" over time:

- Rebuilds (omitting details)
- Multiple extra "seed points" from prev assignment

Often also improve:

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



# Main Improvements: Beating Black-Box HNSW

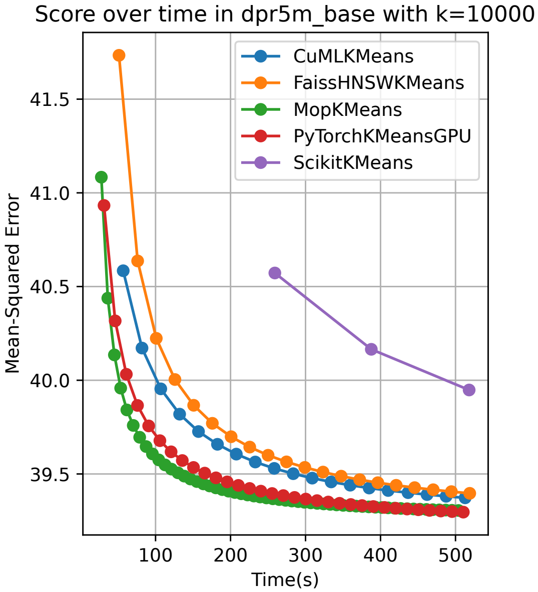
Centroids "slow down" over time:

- Rebuilds (omitting details)
- Multiple extra "seed points" from prev assignment

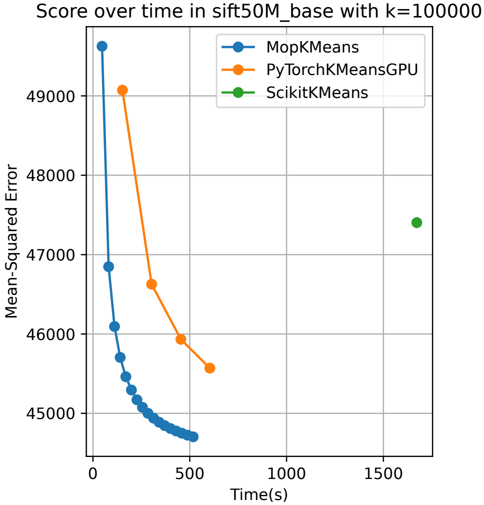
Often also improve:

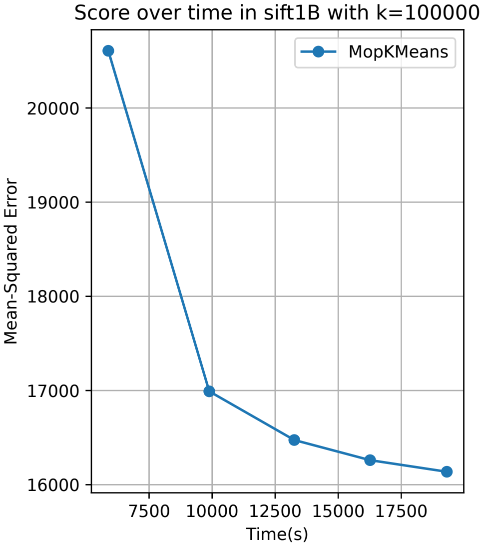
- Min iteration threshold
- Bulk queries for more seed points

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



# More Results

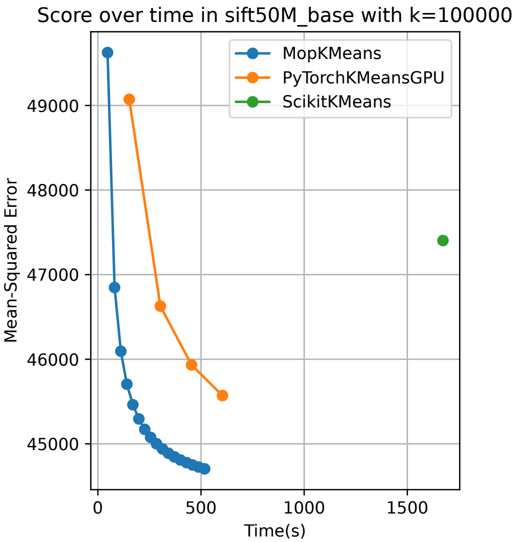


# Conclusion + Open Theory Question

Is this algorithm, or a similar one, directly comparable to some variant of Lloyd's algorithm?

A one-line algorithm change gives good starting point[1] (tad worse in practice, better in theory?)

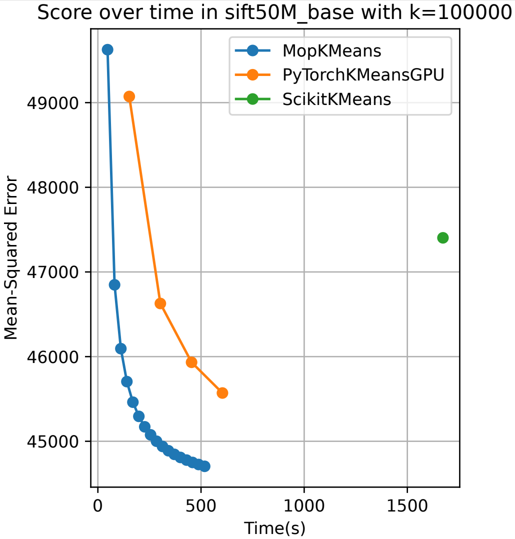



[1] Indyk & Xu, *Worst-case performance of popular approximate nearest neighbor search implementations [...]*

# Conclusion + Open Theory Question

Is this algorithm, or a similar one, directly comparable to some variant of Lloyd's algorithm?

A one-line algorithm change gives good starting point[1] (tad worse in practice, better in theory?)




**Fin.**

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