Algorithm Engineering for High-Dimensional Similarity Search Problems

Martin Aumüller

IT University of Copenhagen

Roadmap

01

Similarity Search in High-Dimensions: Setup/Experimental Approach

02

Survey of state-of-the-art Nearest Neighbor Search algorithms

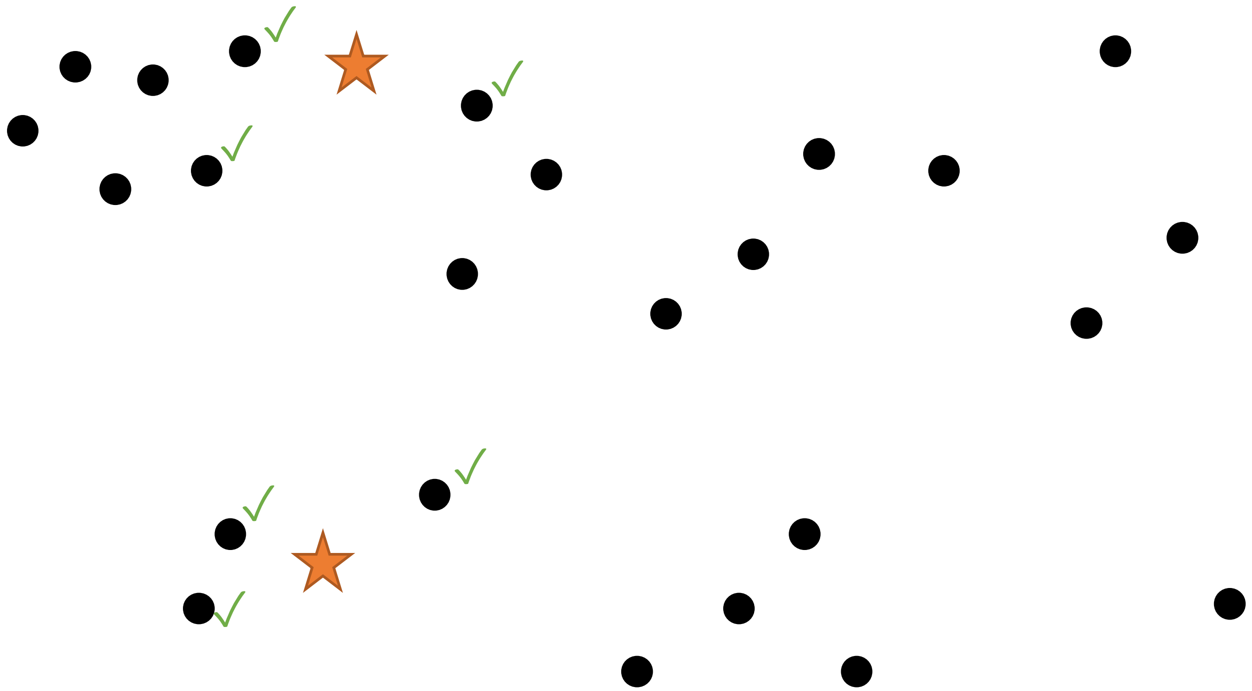
03

Similarity Search on the GPU, in external memory, and in distributed settings

1. Similarity Search in High-Dimensions: Setup/Experimental Approach

k-Nearest Neighbor Problem

- **Preprocessing**: Build DS for set S of n data points
- **Task**: Given query point q, return k closest points to q in S



Nearest neighbor search on words

- GloVe: learning algorithm to find vector representations for words
- *GloVe.twitter* dataset: **1.2M words**, vectors trained from **2B tweets**, **100 dimensions**
- Semantically similar words: nearest neighbor search on vectors

https://nlp.stanford.edu/projects/glove/

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. [GloVe: Global Vectors for Word Representation](https://nlp.stanford.edu/projects/glove/).

GloVe Examples

```
$ grep -n "sicily" glove.twitter.27B.100d.txt
118340:sicily -0.43731 -1.1003 0.93183 0.13311 0.17207 …
```

“sicily”

- sardinia
- tuscany
- dubrovnik
- liguria
- naples

“algorithm”

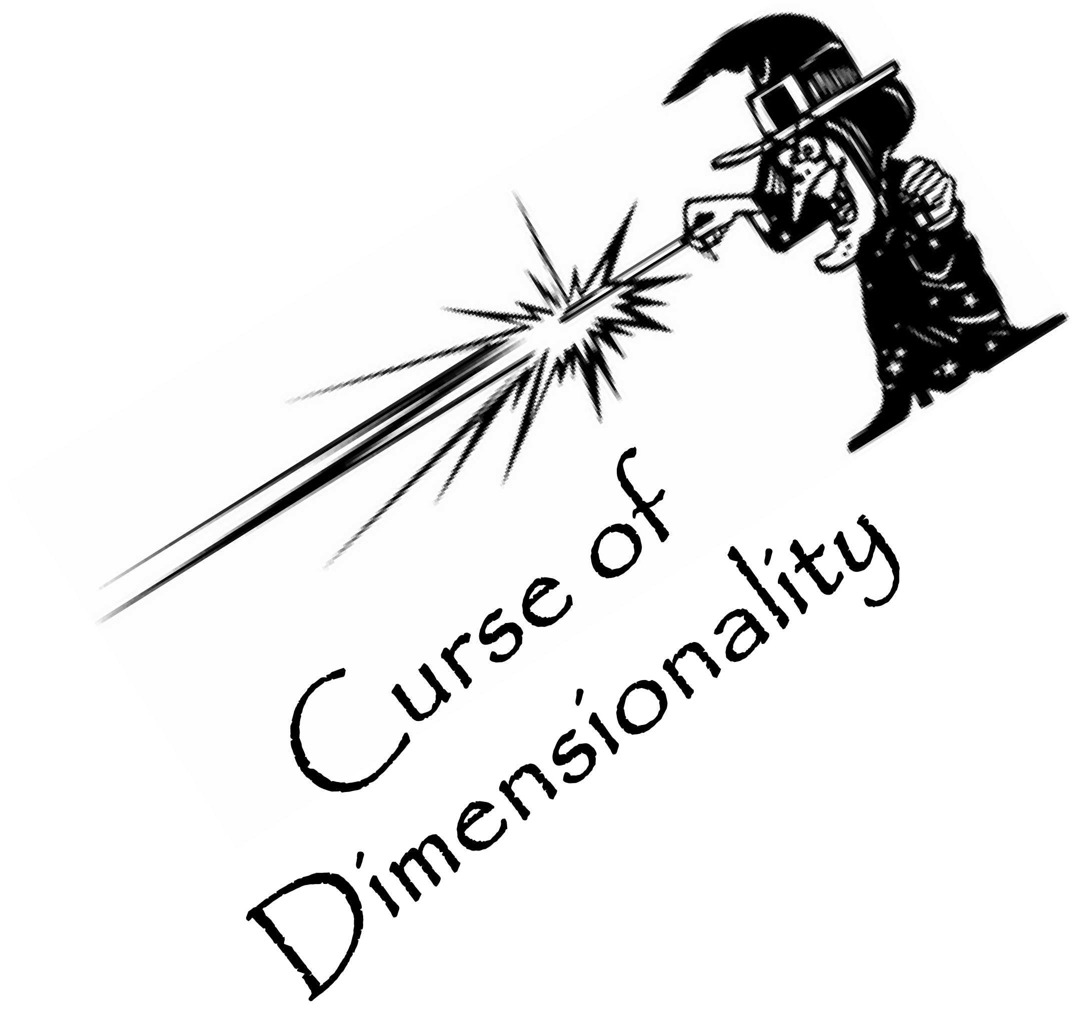
- algorithms
- optimization
- approximation
- iterative
- computation

“engineering”

- engineer
- accounting
- research
- science
- development

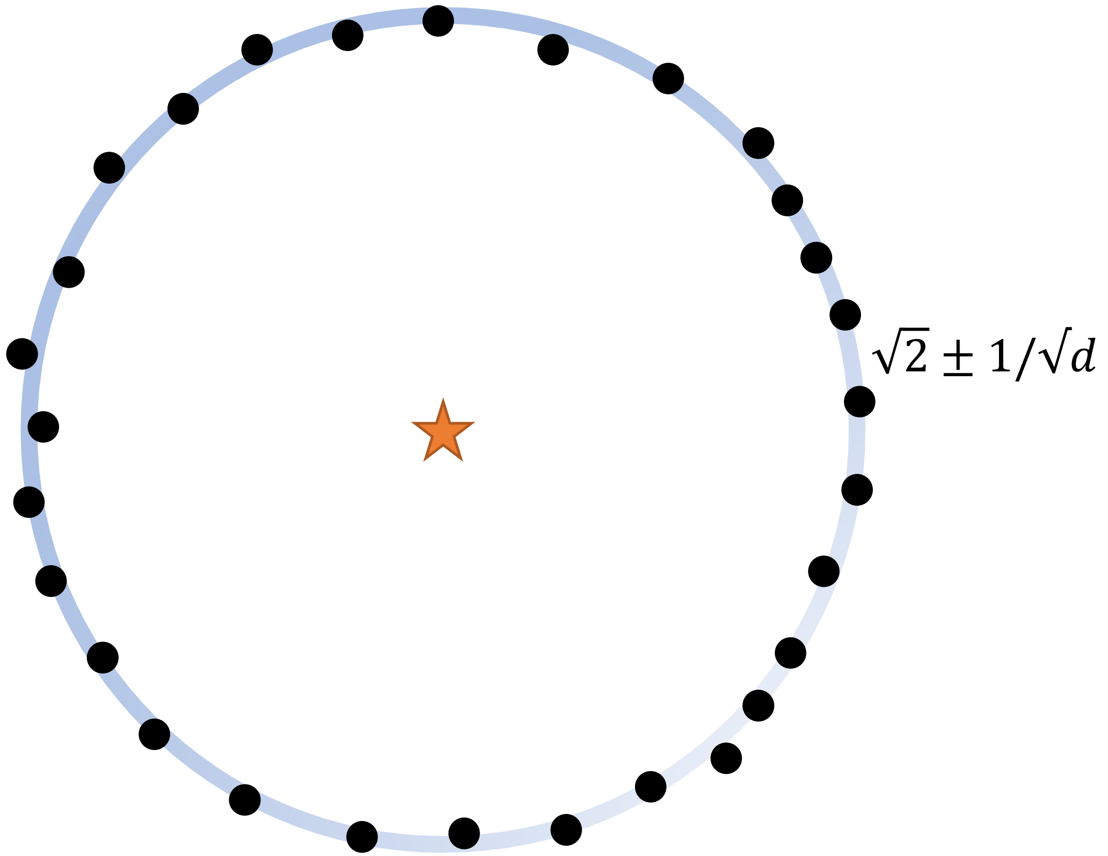
Basic Setup

- Data is described by **high-dimensional feature vectors**
- **Exact similarity search is difficult** in high dimensions
- data structures and algorithms suffer
  - **exponential dependence** on dimensionality
  - **in time, space, or both**



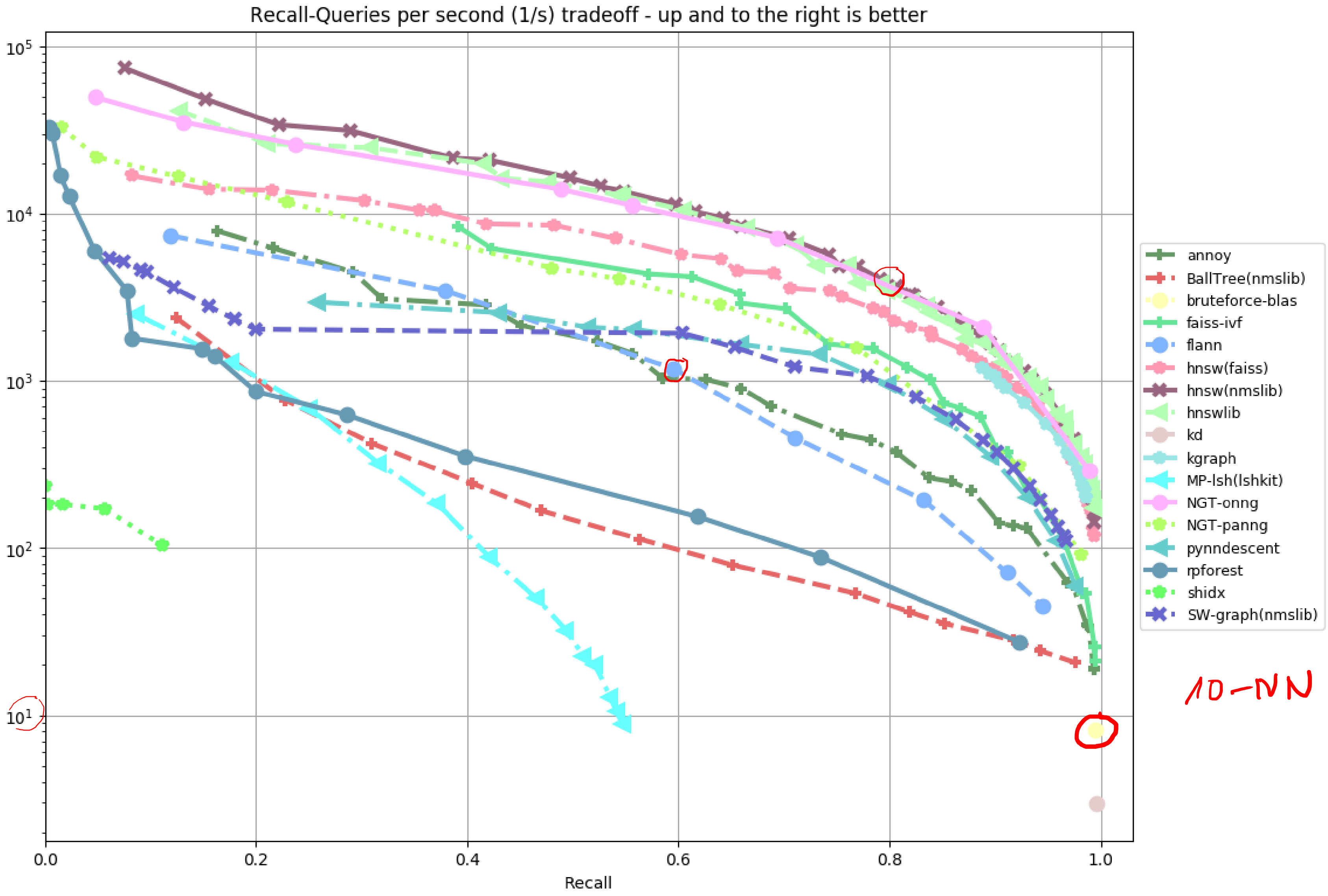
Why is Exact NN difficult?

- Choose n random points from $N(0, 1/d)^d$, for large d
- Choose a random query point



- nearest and furthest neighbor basically at same distance http://ann-benchmarks.com

[A., Bernhardsson, Faithfull, 2020]



Performance on GloVe

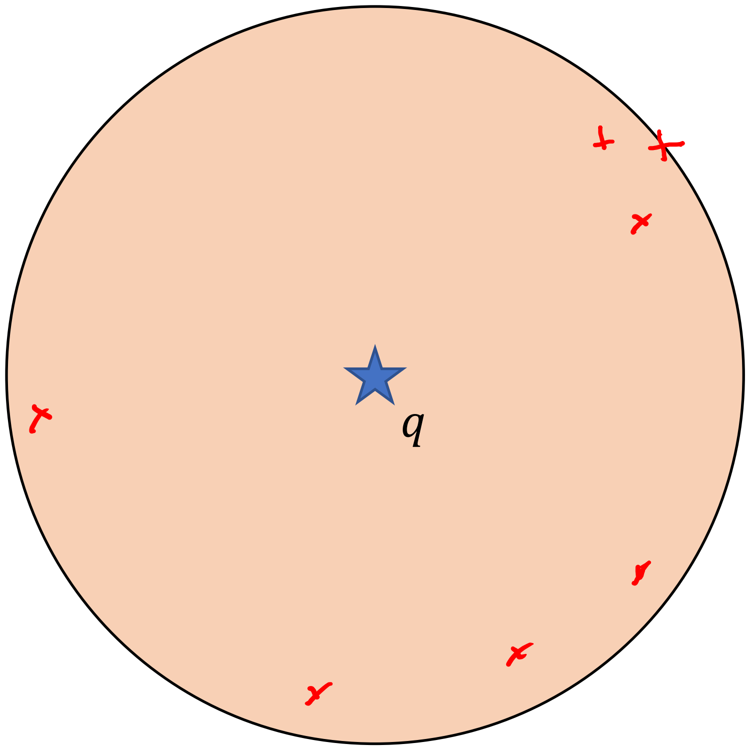
10,000 queries

10-NN

Difficulty measure for queries

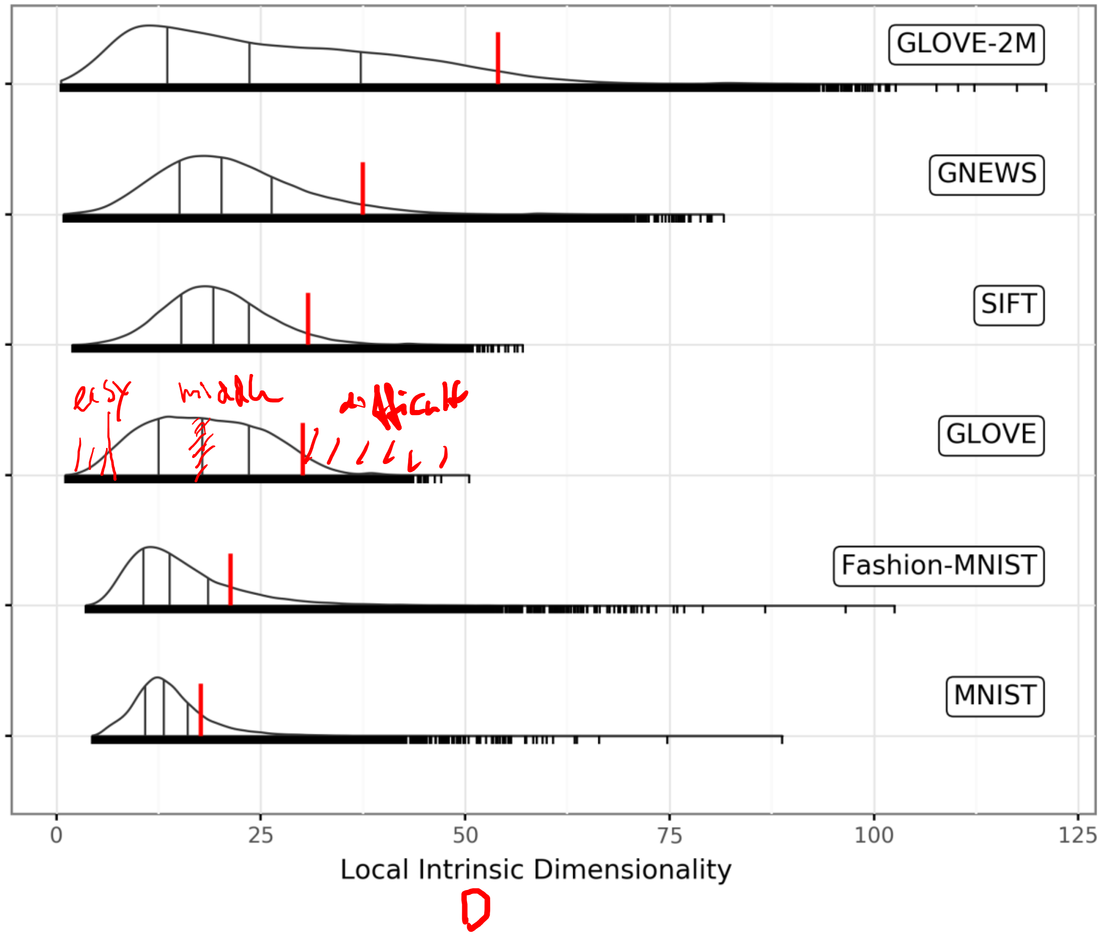
- Given query q and distances $r_1, \ldots, r_k$ to k nearest neighbors, define

$$D(q) = -\left(\frac{1}{k}\sum \ln r_i/r_k\right)^{-1}$$



Based on the concept of local intrinsic dimensionality [Houle, 2013] and its MLE estimator [Amsaleg et al., 2015]

LID Distribution



Dataset	Data Points	Dimensions
SIFT [9]	1 000 000	128
MNIST	65 000	784
Fashion-MNIST [19]	65 000	784
GLOVE [17]	1 183 514	100
GLOVE-2M [17]	2 196 018	300
GNEWS [16]	3 000 000	300

Results (GloVe, 10-NN, 1.2M points)

[A., Ceccarello, 2019]

http://ann-benchmarks.com/sisap19/faiss-ivf.html

2. STATE-OF-THE-ART NEAREST NEIGHBOR SEARCH

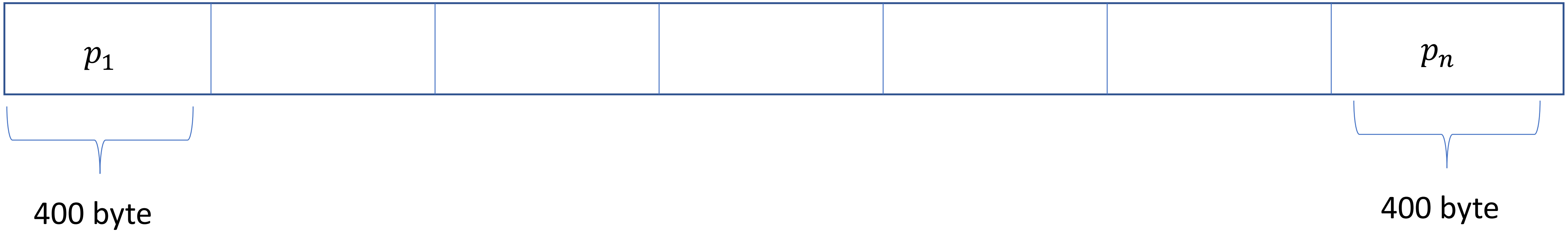
General Pipeline

Index generates candidates

Brute-force search on candidates

Brute-force search

GloVe: 1.2 M points, inner product as distance measure

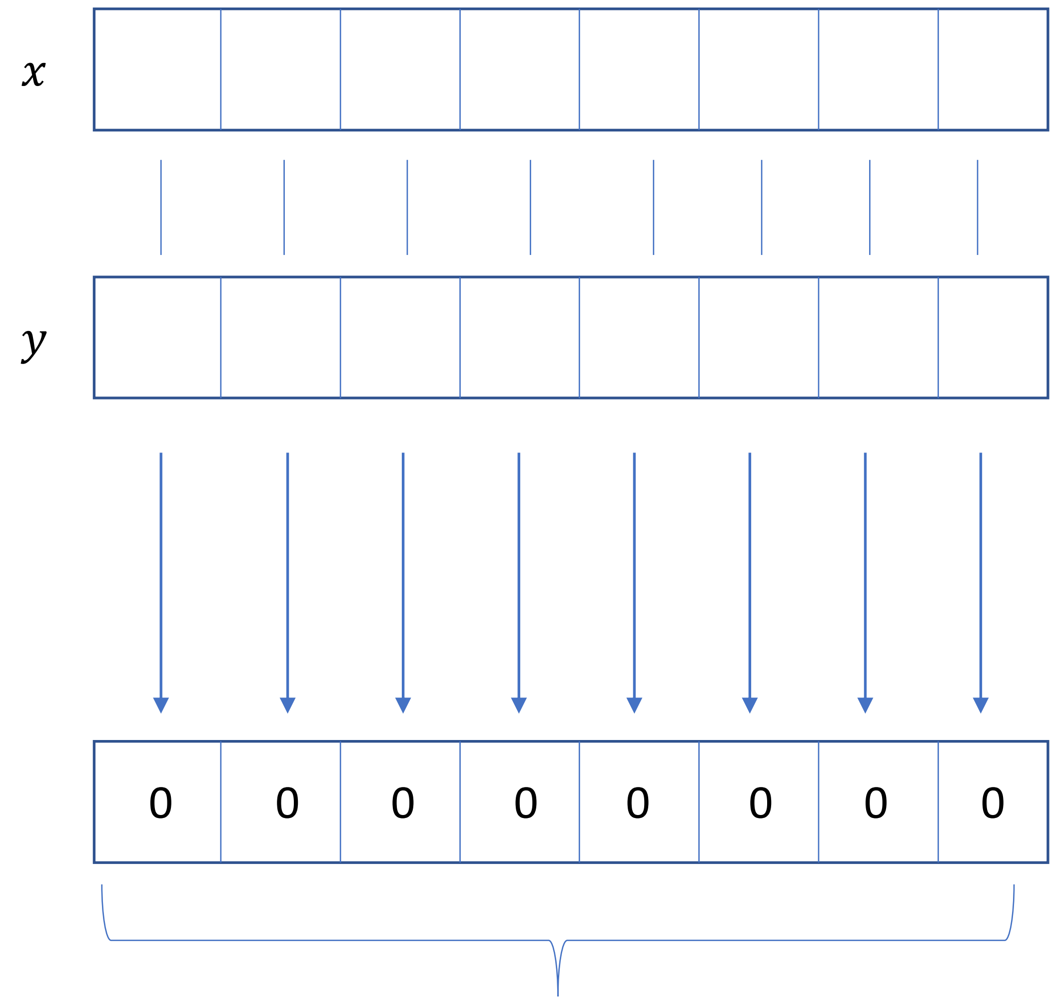



```c
inline float dot_naive(const float* x, const float* y, int f) {
  float result = 0;
  for (int i = 0; i < f; i++) {
    result += x[i] * y[i];
  }
  return result;
}
```

- 100ms per scan
- 4.2 GB/s throughput
- **CPU-bound**

Automatically SIMD vectorized with clang –O3: https://godbolt.org/z/TJX68s https://gist.github.com/maumueller/720d0f71664bef694bd56b2aeff80b17

Manual vectorization (256 bit registers)



…

Parallel multiply

…

Parallel add to result register

Horizontal sum and cast to float

- 25 ms per query
- 16 GB/s
- 16.5 GB/s single-thread max on my laptop
- **Memory-bound**

Brute-force on bit vectors

- Another popular distance measure is Hamming distance
  - Number of positions in which two bit strings differ

- Can be nicely packed into 64-bit words
- Hamming distance of two words is just bitcount of the XOR

```
inline uint64_t distance(const uint64_t* x, const uint64_t* y, int f) {
  uint64_t res = 0;
  for (int i = 0; i < f; i++) {
    res +=  __builtin_popcountll(x[i] ^ y[i]);
  }
  return res;
}
```

- 1.3 ms per query (128 bits)
- 6 GB/s throughput

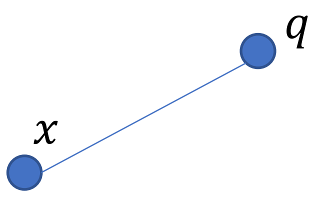
[Christiani, 2019]

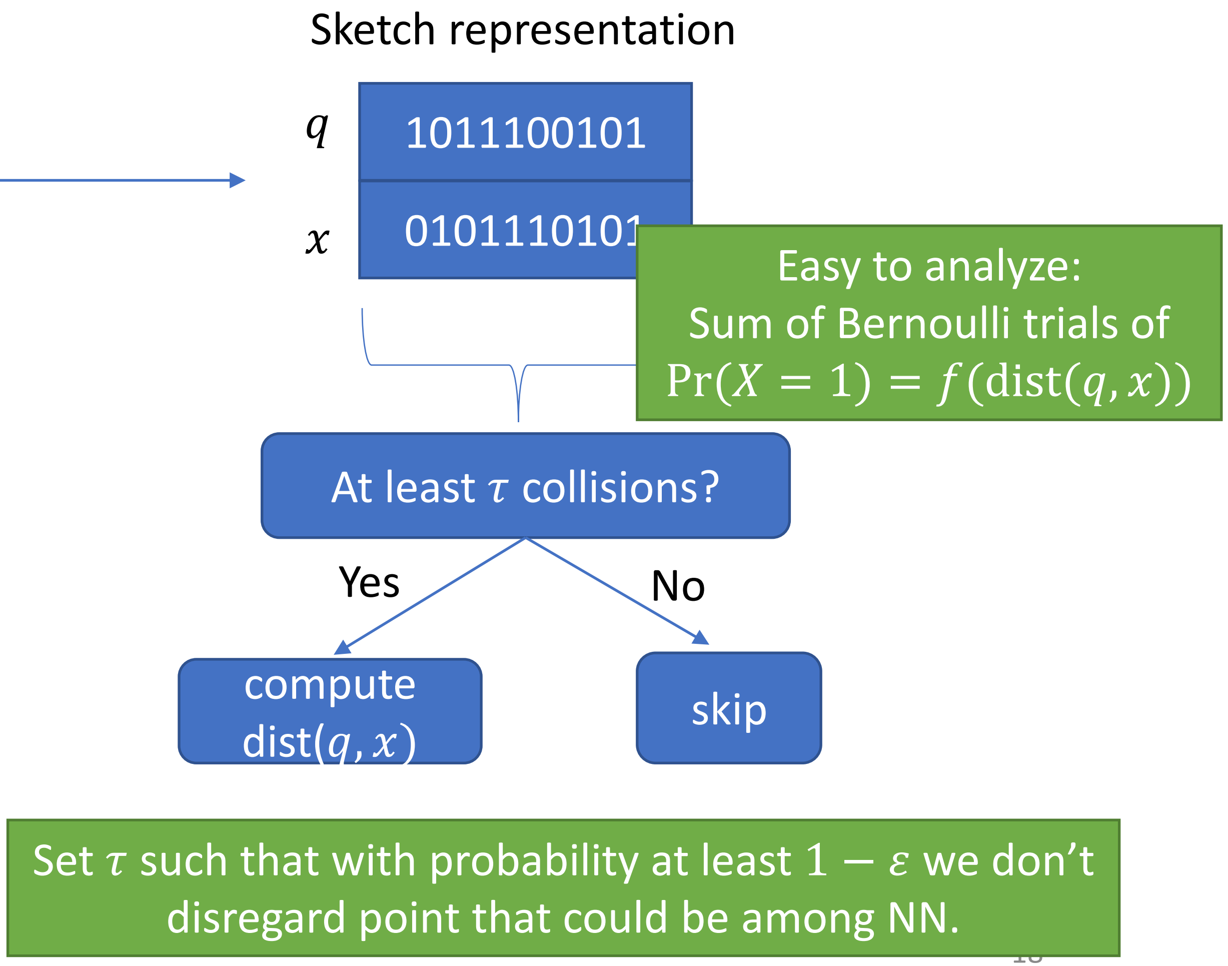
Sketching to avoid distance computations

- Distance computations on bit vectors faster than Euclidean distance/inner product
- Their number can be reduced by storing compact sketch representations

SimHash [Charikar, 2002]
1-BitMinHash [König-Li, 2010]

Can distance computation be avoided?





General Pipeline

Index generates candidates

Brute-force search on candidates

Credit: Richard Bartz

PUFFINN

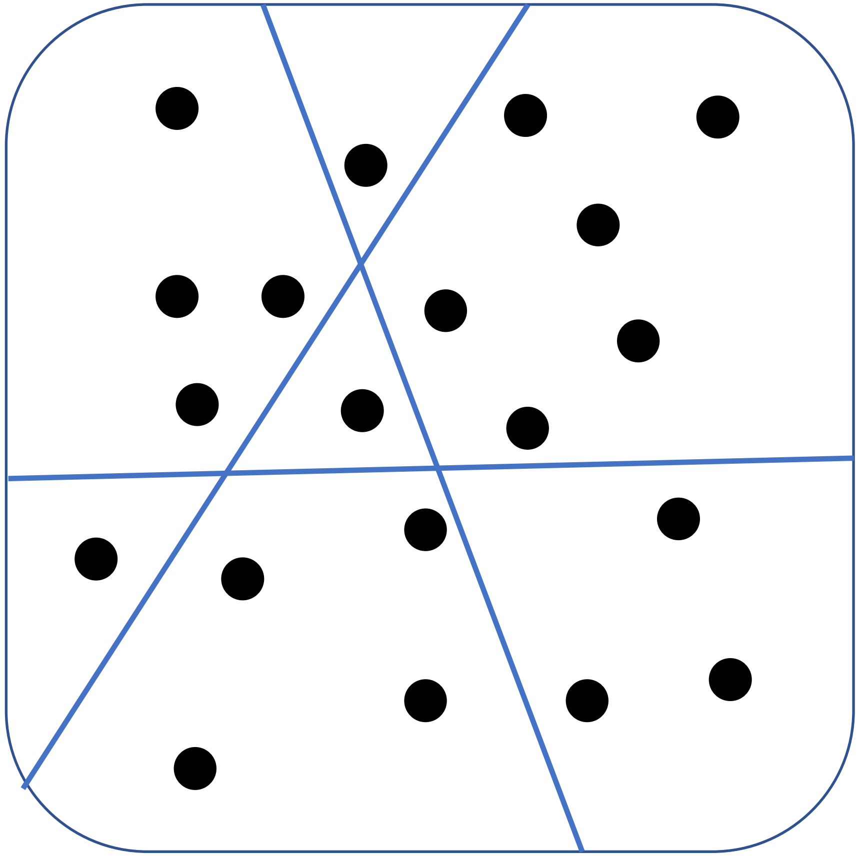
PARAMETERLESS
AND **U**NIVERSALLY
FAST **FI**NDING OF
NEAREST
NEIGHBORS

[A., Christiani, Pagh, Vesterli, 2019]

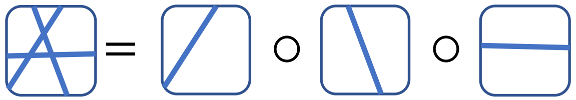
https://github.com/puffinn/puffinn

How does it work?

Locality-Sensitive Hashing (LSH) [Indyk-Motwani, 1998]

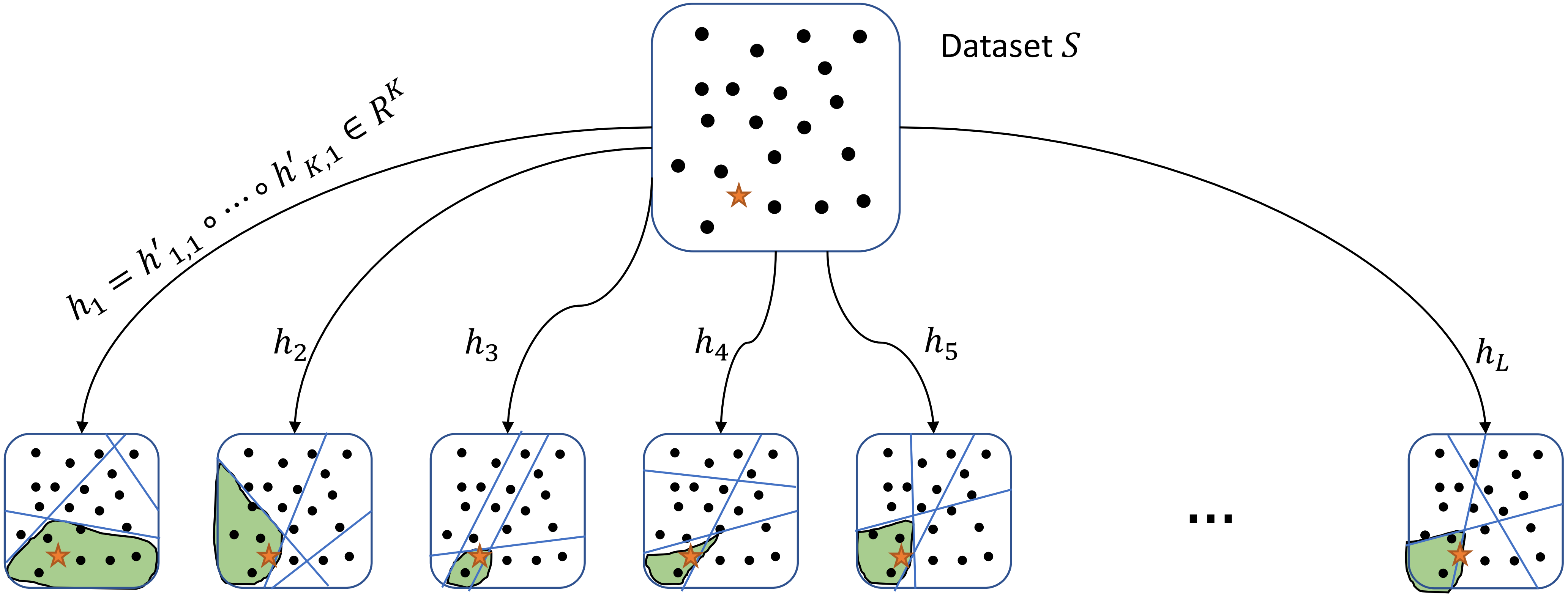


$h(p) = h_1(p) \circ h_2(p) \circ h_3(p) \in \{0,1\}^3$



A family $\mathcal{H}$ of hash functions is **locality-sensitive**, if the collision probability of two points is decreasing with their distance to each other.

Solving k-NN using LSH (with failure prob. δ)



Termination: If $(1-p)^j \leq \delta$, report **current** top-k.

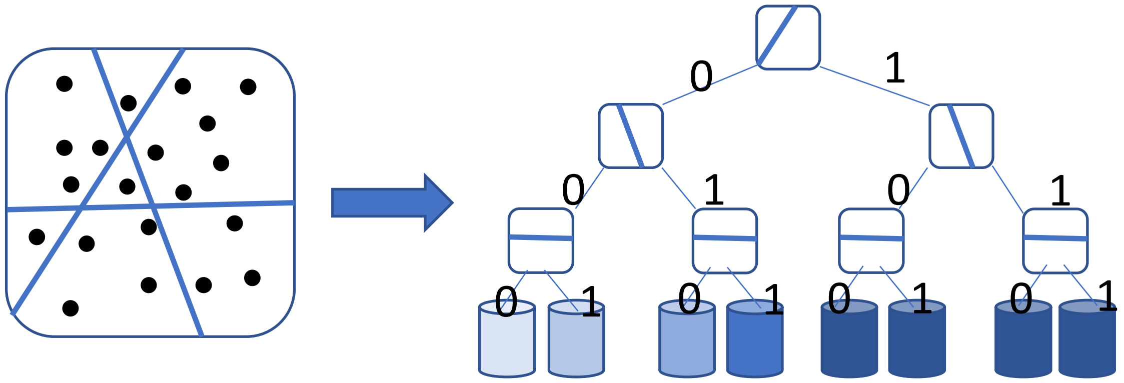
probability of the **current** k-th nearest neighbor to collide.

Not terminated? Decrease K!

The Data Structure

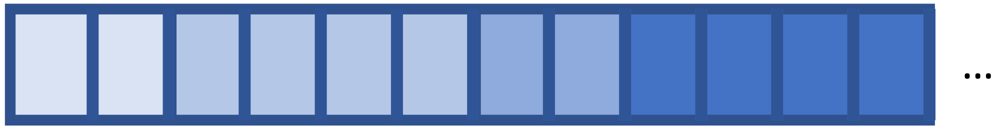
Theoretical

- **LSH Forest:** Each repetition is a Trie build from LSH hash values [Bawa et al., 2005]

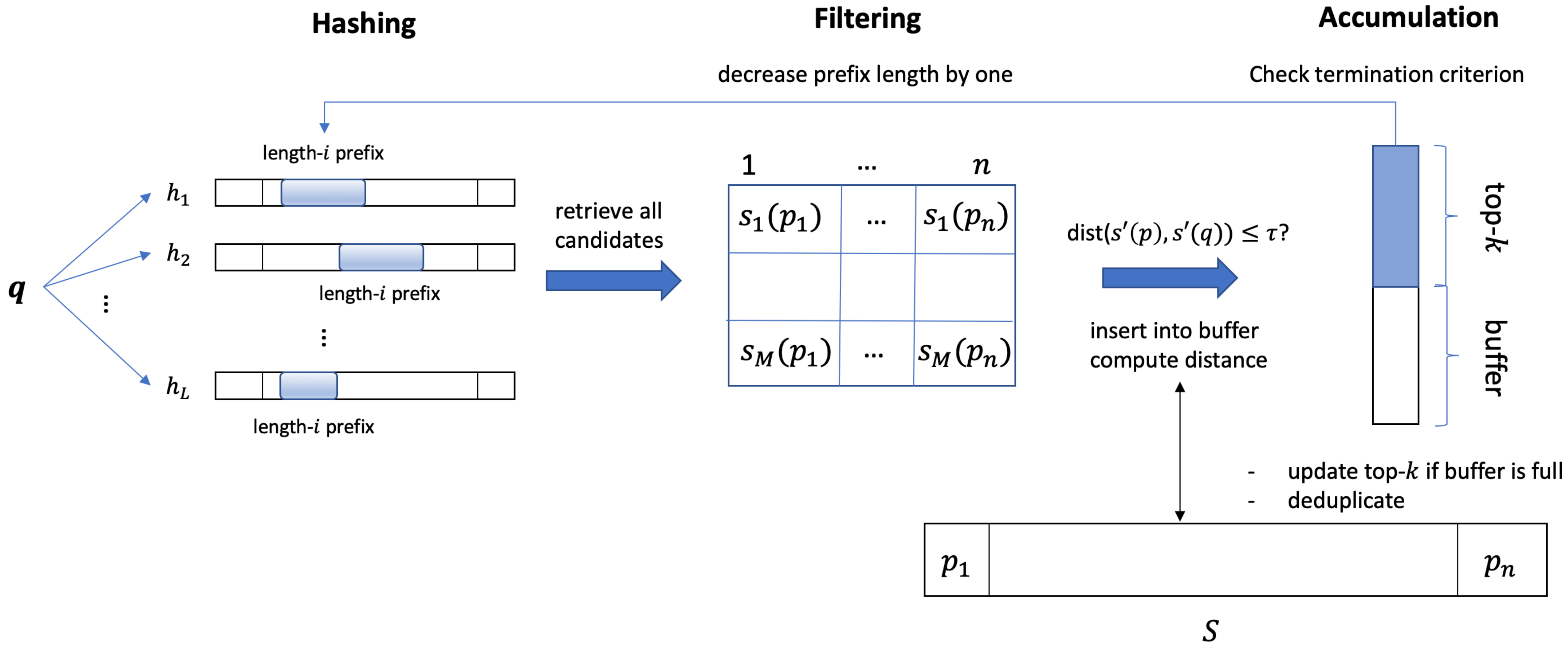


Practical

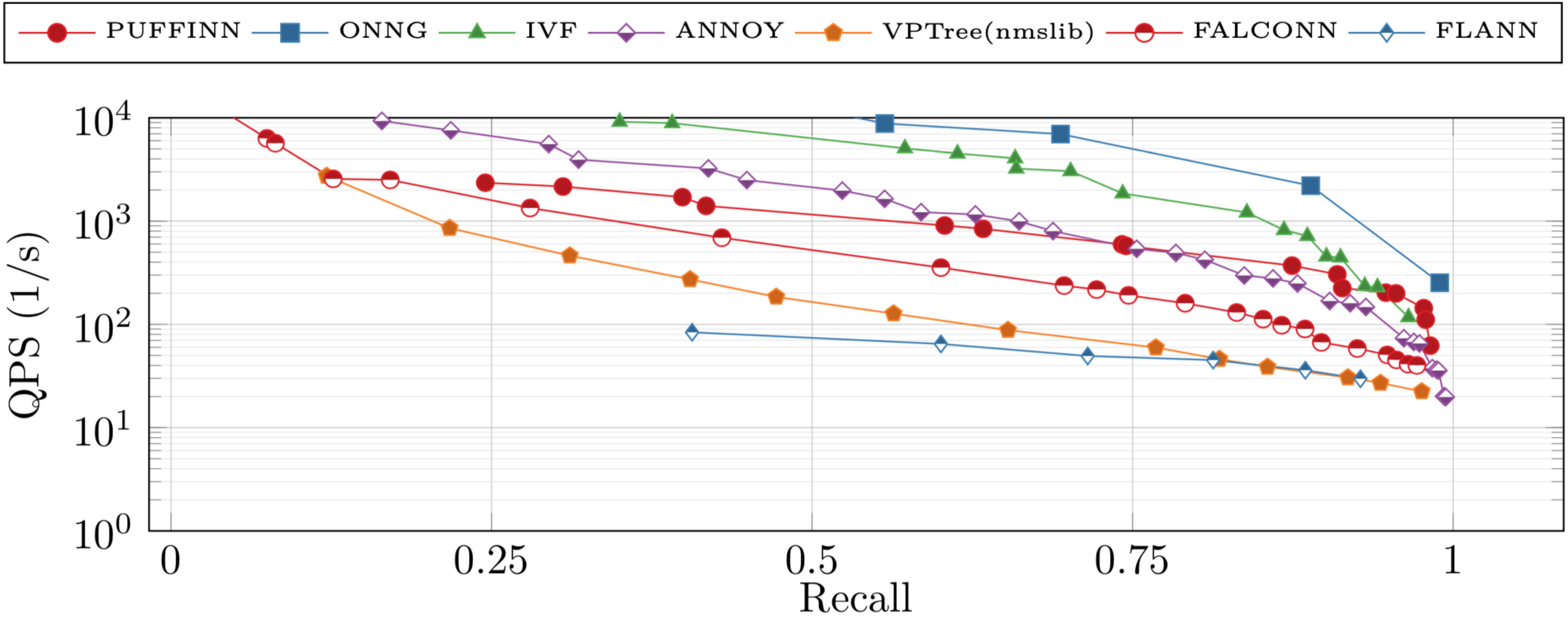
- Store indices of data set points sorted by hash code
- ”Traversing the Trie” by binary search
- use lookup table for first levels



Overall System Design



Running time (Glove 100d, 1.2M, 10-NN)



A difficult (?) data set in $\mathbb{R}^{3d}$

n data points

$$x_1 = (0^d, y_1, z_1)$$

$$\vdots$$

$$x_{n-1} = (0^d, y_{n-1}, z_{n-1})$$

$$x_n = (v, w, 0^d)$$

m query points

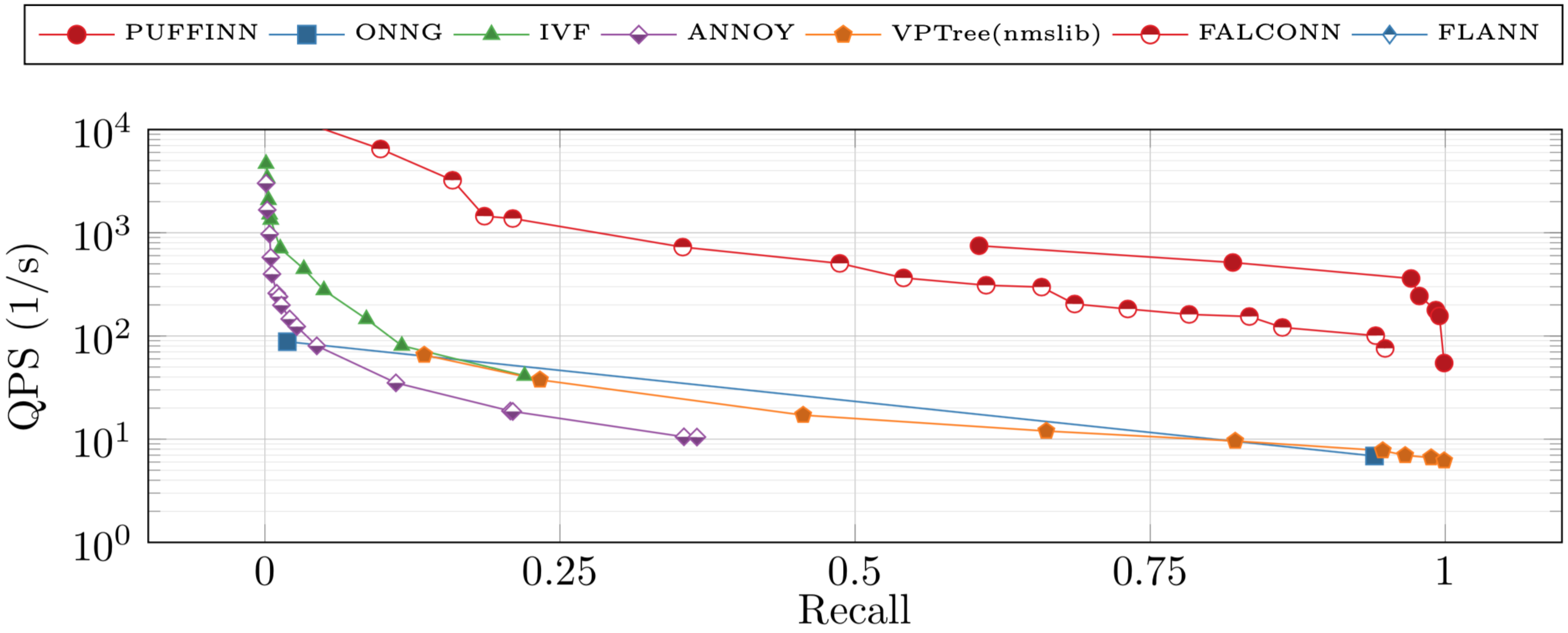
$$q_1 = (v, 0^d, r_1)$$

$$\vdots$$

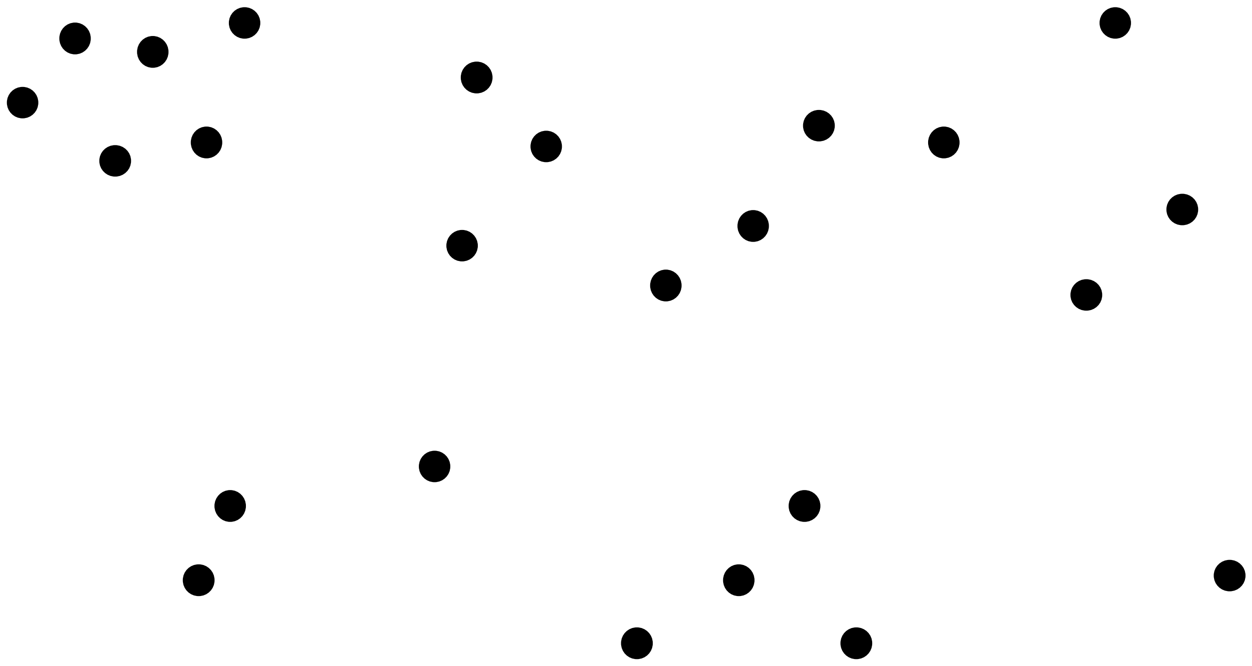
$$q_m = (v, 0^d, r_m)$$

$$y_i, z_i, v, w, r_i \sim \mathcal{N}^d\left(0, \frac{1}{2d}\right)$$

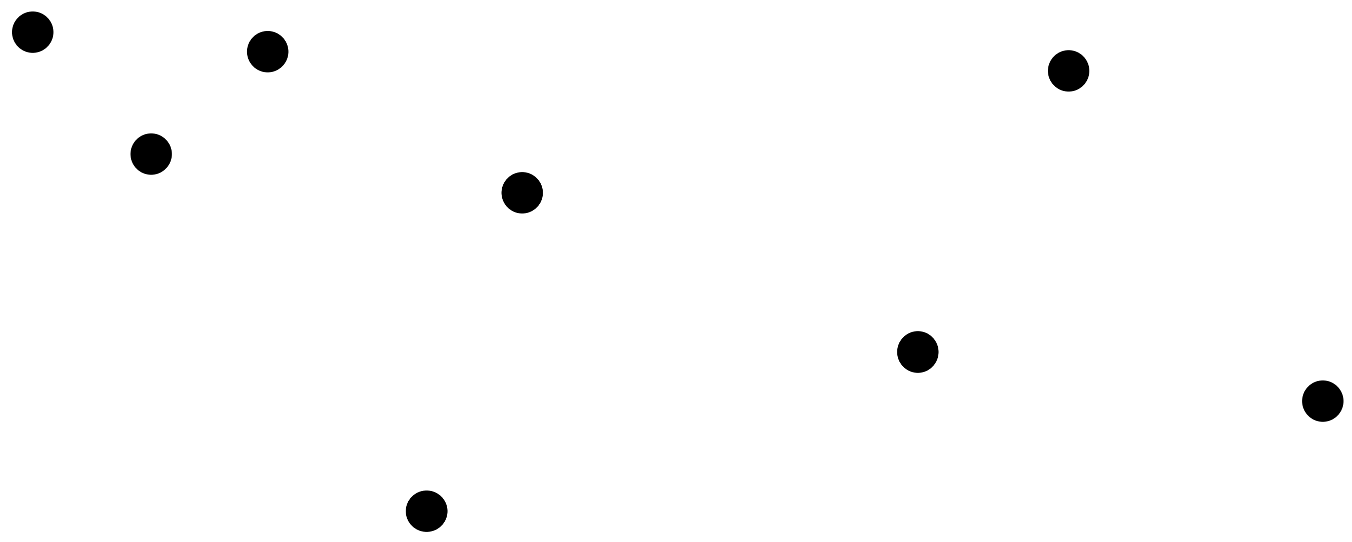
Running time (“Difficult”, 1M, 10-NN)



Graph-based Similarity Search

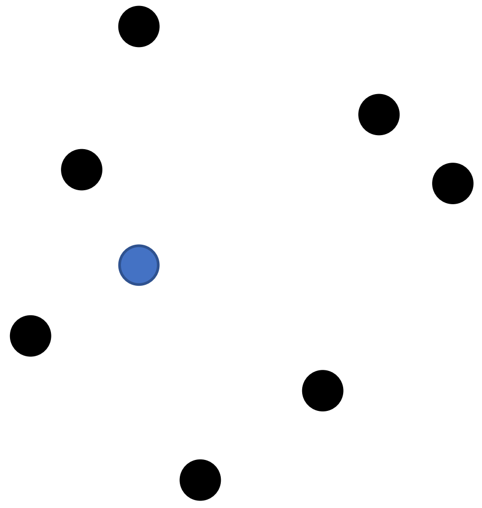


Building a Small World Graph



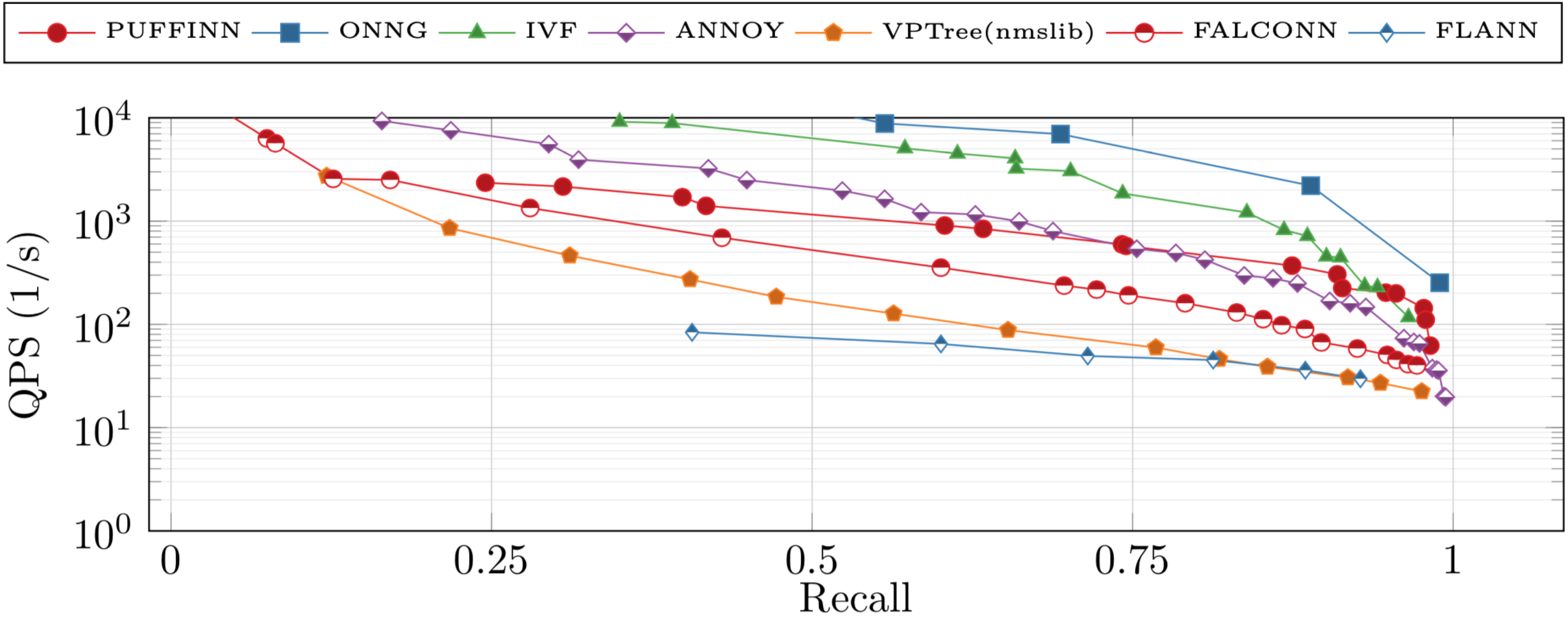
Refining a Small World Graph

Goal: Keep out-degree as small as possible (while maintaining “large-enough” in-degree)!



HNSW/ONNG: [Malkov et al., 2020], [Iwasaki et al., 2018]

Running time (Glove 100d, 1.2M, 10-NN)



Open Problems Nearest Neighbor Search

- Data-dependent LSH with guarantees?
- Theoretical sound Small-World Graphs?
- Multi-core implementations
  - Good? [Malkov et al., 2020]
- Alternative ways of sketching data?

3. Similarity Search on the GPU, in External Memory, and in Distributed Settings

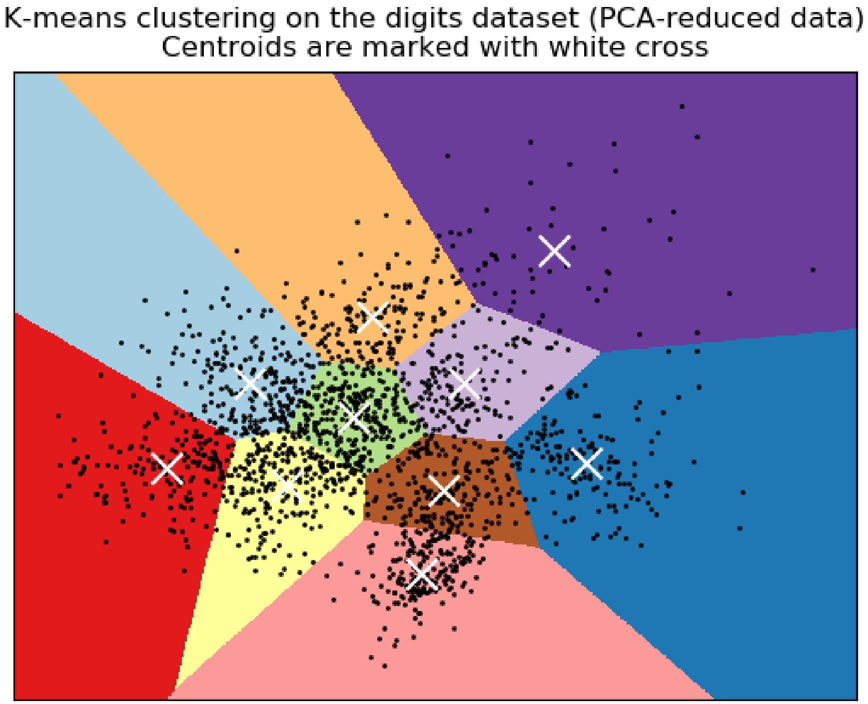
Nearest Neighbors on the GPU: FAISS

[Johnson et al., 2017] https://github.com/facebookresearch/faiss

- GPU setting
  - Data structure is held in GPU memory
  - Queries come in batches of say 10,000 queries per time
- Results:
  - http://ann-benchmarks.com/sift-128-euclidean_10_euclidean-batch.html

FAISS/2

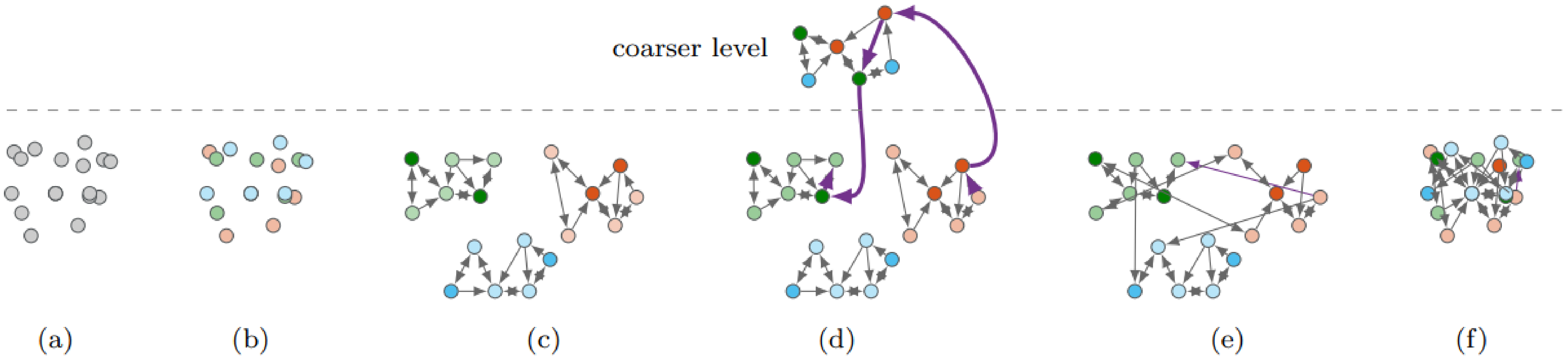
- Data structure
  - Run k-means with large number of centroids
  - Each data point is associated with closest centroid
- Query
  - Find L closest centroids
  - Return k closest points found in points associated with these centroids



https://scikit-learn.org/stable/auto_examples/cluster/plot_kmeans_digits.html

Nearest Neighbors on the GPU: GGNN

[Groh et al., 2019]



Nearest Neighbors in External Memory

[Subramanya et al., 2019]

RAM

$\widehat{x_1}$

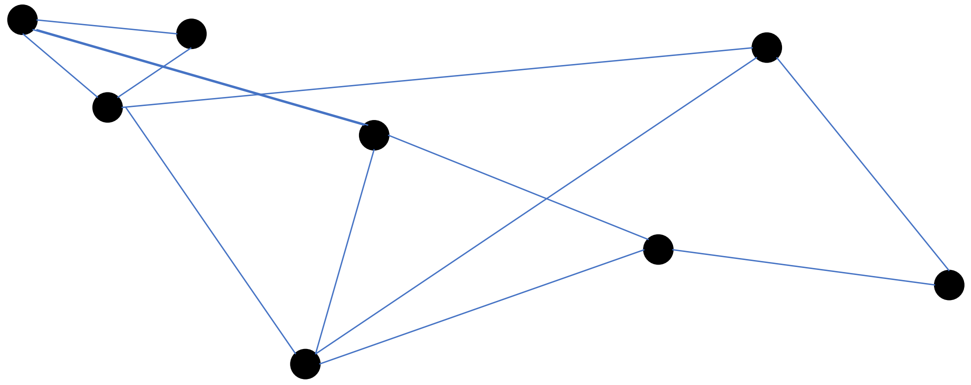
⋮

$\widehat{x_n}$

compressed vectors
(32 byte per vector)

Product Quantization

SSD



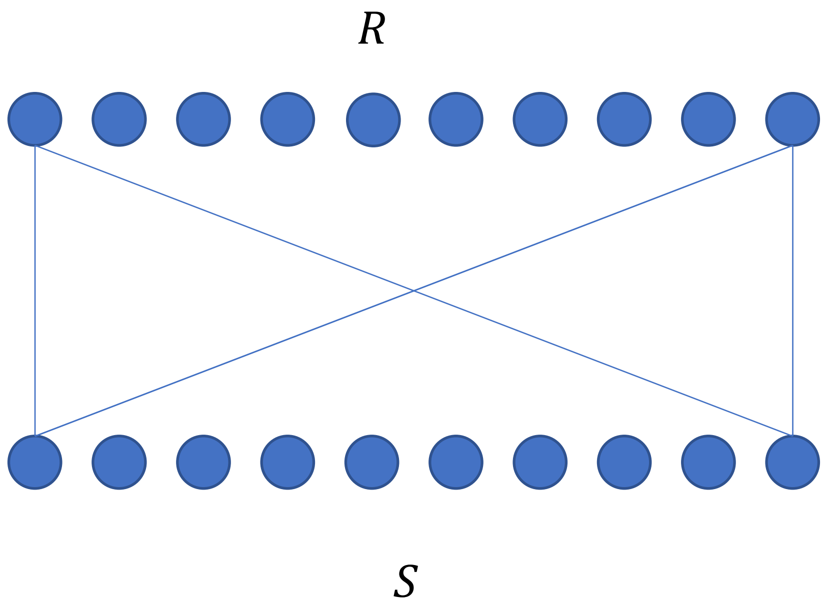
x_1 … x_n

Original vectors
(~400 byte per vector)

Distributed Setting: Similarity Join

- Problem
  - given sets R and S of size n,
  - and similarity threshold λ, compute

$$R \bowtie_\lambda S = \{(x,y) \in R \times S \mid sim(x,y) \geq \lambda\}$$



- Similarity measures
  - Jaccard similarity
  - Cosine similarity

- Naive: $O(n^2)$ distance computations

Map-Reduce-based Similarity Join

Single Core on Xeon E5-2630v2 (2.60 GHz)

LIVEJ	PPJ 345 PEL	PEL 88.9 PPJ	PEL 22.1 PPJ	PEL 12.0 PPJ GRP	PEL 6.52 PPJ GRP MPJ	PPJ 3.50 GRP PEL MPJ ALL PP+	MPJ 1.88 ALL PEL PPJ GRP PP+	ALL 1.02 MPJ PEL
NETFLIX	ALL 1235 PPJ GRP	ALL 494 ADP PPJ GRP	ADP 146	ADP 76.4	ADP 36.6	ADP 15.6 PPJ GRP	PPJ 4.73 GRP	PPJ 0.894 ALL PEL GRP MPJ
ORKUT	PPJ 213 GRP	PPJ 79.4 GRP	PPJ 33.4 GRP PEL	GRP 21.0 PPJ PEL MPJ	GRP 12.9 PPJ MPJ PEL	MPJ 7.69 GRP PPJ PEL	MPJ 4.28 ALL PEL GRP	MPJ 2.06 ALL PEL PP+

[Mann et al., 2016]

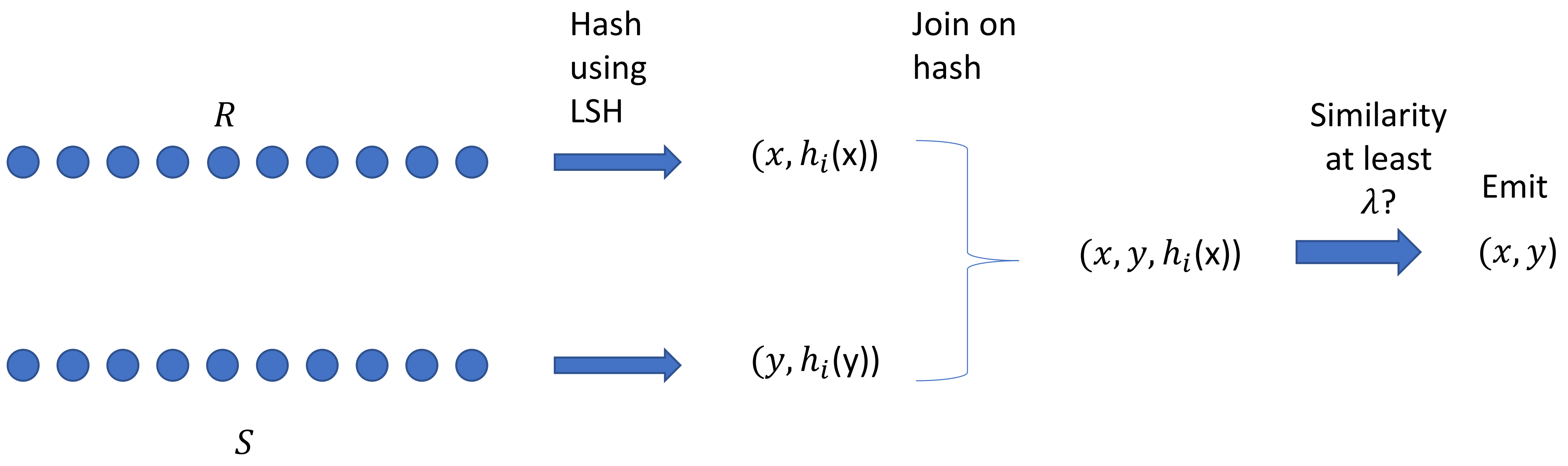
Hadoop cluster (12 nodes, 24 HT per node)

LIVE	313 VJ	285 VJ	278 VJ MG	254 VJ MG	243 VJ GJ, MG
NETF	T	T	527 VJ	215 VJ	161 VJ
ORKU	T	1592 MG	941 VJ MG	761 GJ VJ	681 VJ

[Fier et al., 2018]

Solved almost-optimally in the MPC model

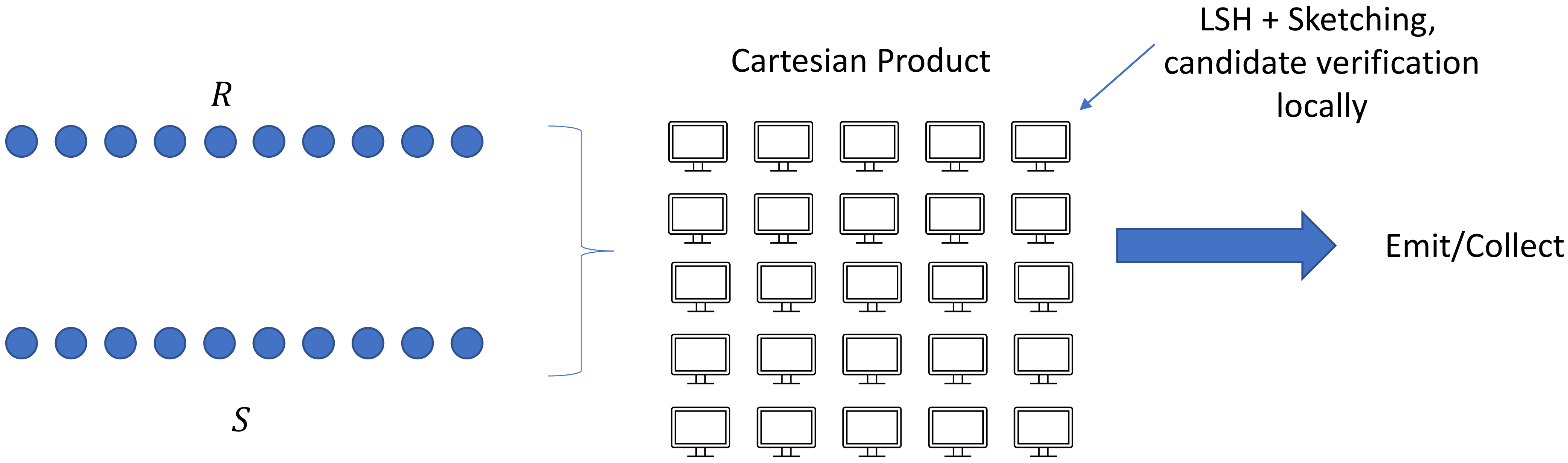
[Hu et al., 2019]



$O(n^2)$ local work for distance computations!

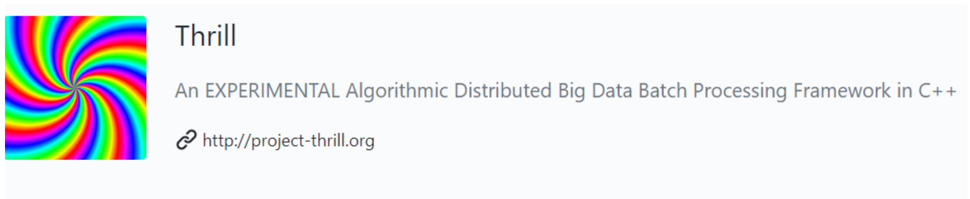
Another approach: DANNY

[A., Ceccarello, Pagh, 2020] In preparation, https://github.com/cecca/danny

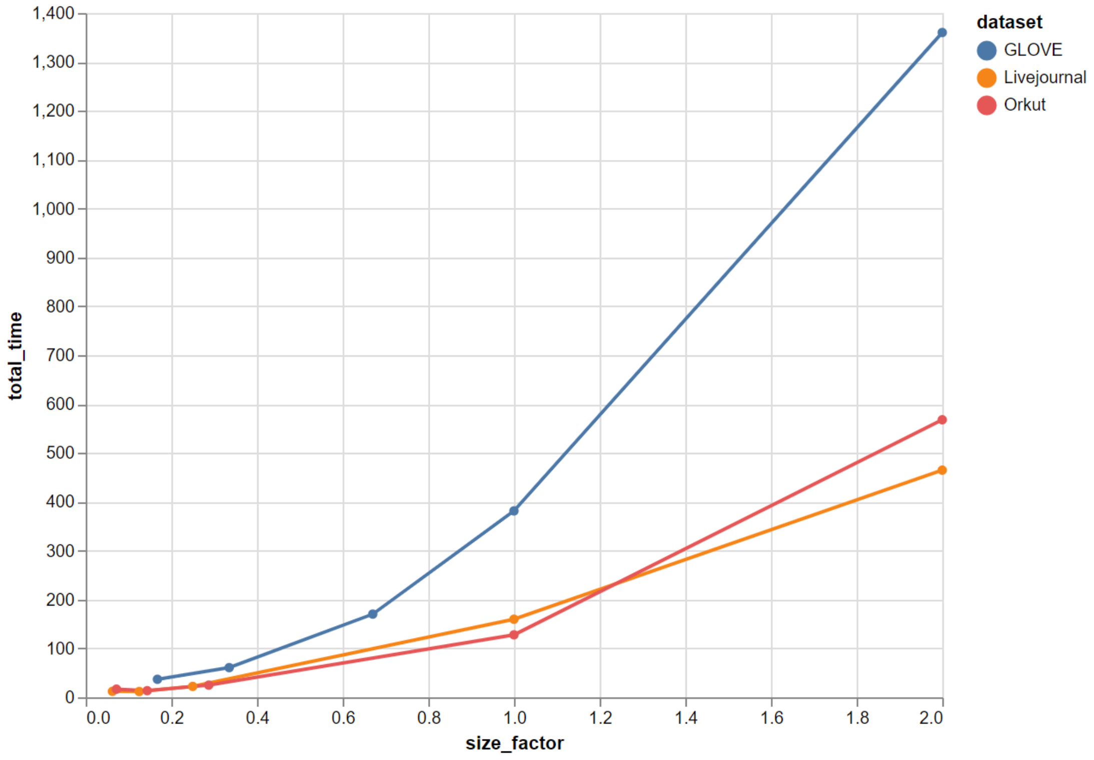


Implementation in Rust using timely dataflow

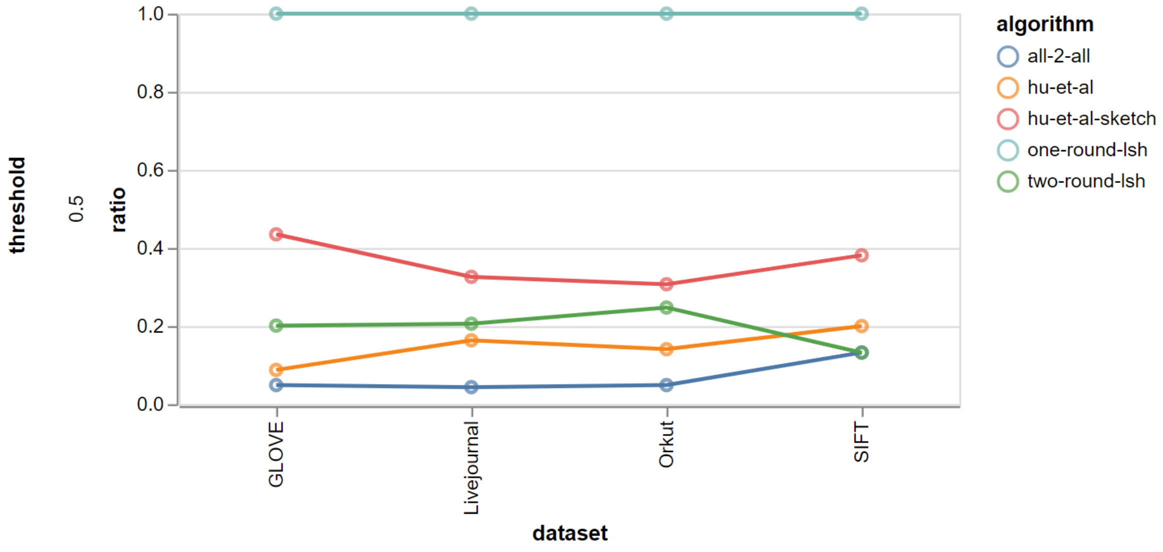
https://github.com/TimelyDataflow/timely-dataflow



Results



1,400
1,300
1,200
1,100
1,000
900
800
700
600
500
400
300
200
100
0
total_time
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
1.6
1.8
2.0
size_factor
dataset
GLOVE
Livejournal
Orkut



threshold
0.5
ratio
1.0
0.8
0.6
0.4
0.2
0.0
GLOVE
Livejournal
Orkut
SIFT
dataset
algorithm
all-2-all
hu-et-al
hu-et-al-sketch
one-round-lsh
two-round-lsh

Roadmap

01

Similarity Search in High-Dimensions: Setup/Experimental Approach

02

Survey of state-of-the-art Nearest Neighbor Search algorithms

03

Similarity Search on the GPU, in external memory, and in distributed settings

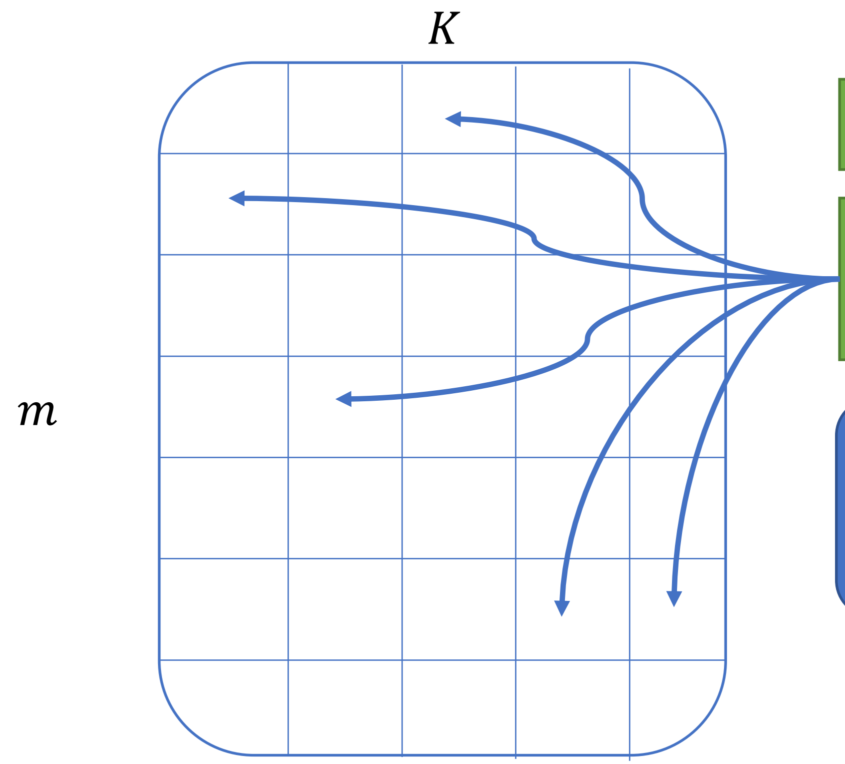
References

- [Amsaleg, Chelly, Furon, Girard, Houle, Kawarabayashi, Nett, 2015]: Estimating local intrinsic dimensionality. KDD 2015.
- [A., Bernhardsson, Faithfull, 2020]: ANN-Benchmarks: A benchmarking tool for approximate nearest neighbor algorithms. Inf. Syst. **87** (2020), see https://arxiv.org/abs/1807.05614 for an open access version.
- [A., Ceccarello, 2019]: The role of local intrinsic dimensionality in benchmarking nearest neighbor search. In: SISAP 2019, see http://ann-benchmarks.com/sisap19/.
- [A., Christiani, Pagh, Vesterli, 2019]: PUFFINN: parameterless and universally fast finding of nearest neighbors. In: ESA 2019, see https://github.com/puffinn/puffinn
- [Christiani, 2019]: Fast locality-sensitive hashing frameworks for approximate near neighbor search
- [Fier, Augsten, Bouros, Leser, Freytag, 2018]: Set similarity joins on MapReduce: An experimental survey. VLDB 2018.
- [Groh, Ruppert, Wieschollek, Lensch, 2019]: GGNN: Graph-based GPU nearest neighbor search https://arxiv.org/abs/1912.01059
- [Houle, 2013]: Dimensionality, discriminability, density and distance distributions. ICDMW 2013.
- [Hu, Yi, Tao, 2019]: Output-optimal massively parallel algorithms for similarity joins. ACM Transactions on Database Systems, 2019.
- [Iwasaki, Miyazaki, 2018]: Optimization of Indexing Based on k-Nearest Neighbor Graph for Proximity Search in High-dimensional Data, https://arxiv.org/abs/1810.07355
- [Indyk, Motwani, 1998]: Approximate Nearest Neighbors: Towards removing the curse of dimensionality, STOC 1998.
- [Johnson, Douze, Jegou, 2017]: Billion-scale similarity search with GPUs, https://arxiv.org/abs/1702.08734.
- [Malkov, Yashunin, 2020]: Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. IEEE TPAM 2020.
- [Mann, Augsten, Bouros, 2016]: An empirical evaluation of set similarity join techniques. VLDB 2016.
- [McSherry, Isard, Murray, 2015], Scalability! But at what COST? USENIX HotOS 2015.
- [Subramanya, Devvrit, Kadekodi, Krishnaswamy, Simhadri, 2019]: DiskANN: Fast accurate billion-point nearest neighbor search on a single node. NeurIPS 2019

Extra slides

PUFFINN: Fast Hash Function Evaluation

- **Main Bottleneck**: Computation of Hash Values
- Adapt the "pooling" technique of `[Dahlgaard et al., 2017]` and `[Christiani, 2019]`



$K \cdot m$ independent hash functions from LSH family, $m \ll L$.

Pick K hash functions in repetition j using universal hash functions in each column.

Analysis using Cantelli's inequality →
Requires different stopping criteria (factor 2 slowdown)