

Data-dependent methods for similarity search in high dimensions

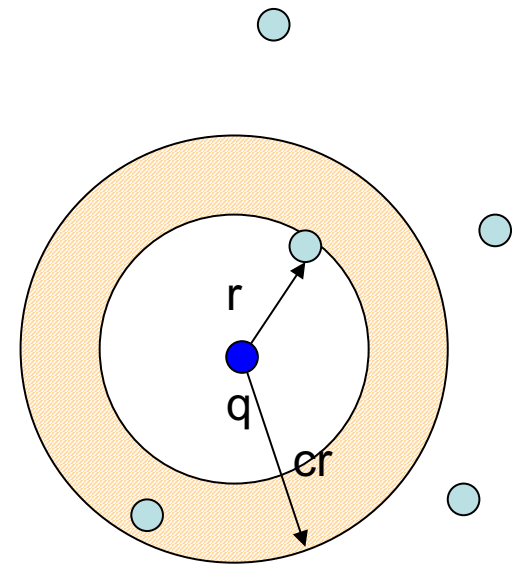
Piotr Indyk
MIT

Plan

- On the benefits of **using data** to design similarity search data structures
- ...and why it took so long to get it to work (provably, for general high-dimensional pointsets)

Problem 1: Approximate Near Neighbor

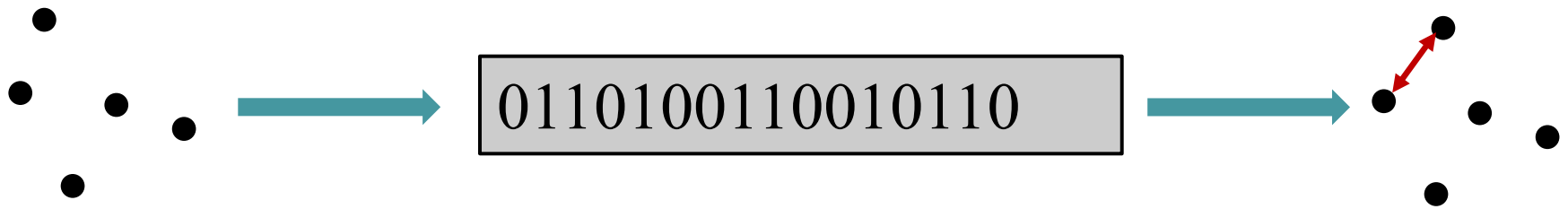
- Given: a set P of n points in a d -dimensional space* and parameters c and r
- Build a data structure which, for any query q :
 - If there is $p \in P$ s.t. $\|q-p\| \leq r$,
 - Then return $p' \in P$ s.t. $\|q-p'\| \leq cr$
- Why approximate ?
 - Exact algorithm with $n^{1-\beta}$ query time and polynomial preprocessing would violate certain complexity-theoretic conjecture (SETH)
 - Reflected in empirical performance, a.k.a., curse of dimensionality
 - Approximate algorithms are typically faster (in theory and practice)
 - E.g., LSH yields $O(dn^{1/c^2})$ query time and $O(dn + n^{1+1/c^2})$ space [Andoni-Indyk'06]



*We assume Euclidean distance in this talk

Problem 2: Approximate Metric Compression

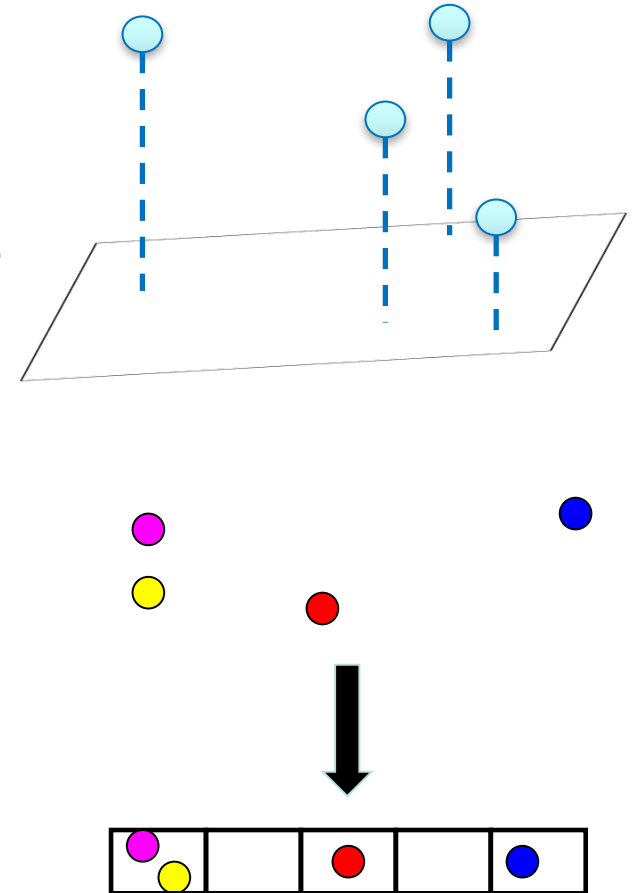
- Given: a set P of n points in a d -dimensional space
- Compress it into as few *bits* as possible
- Estimate pairwise distances within $1+\epsilon$



- Exact representation: $O(ndB)$
 - B =bits of precision; we assume $B=O(\log n)$
- Random projection: $O(n \log(n)/\epsilon^2 B) = O(n \log(n)^2/\epsilon^2)$

What do these problems have in common ?

- Until recently, the fastest provable approaches constructed data structures that were essentially oblivious* to the data
 - Random projection
 - Locality-Sensitive Hashing
- Benefits: flexibility, provability
 - E.g., for distributed data, no need to coordinate between nodes
- Drawbacks: Less efficient



*Formally, the memory cells accessed to answered queries depend on query q but not on P

The “idea”

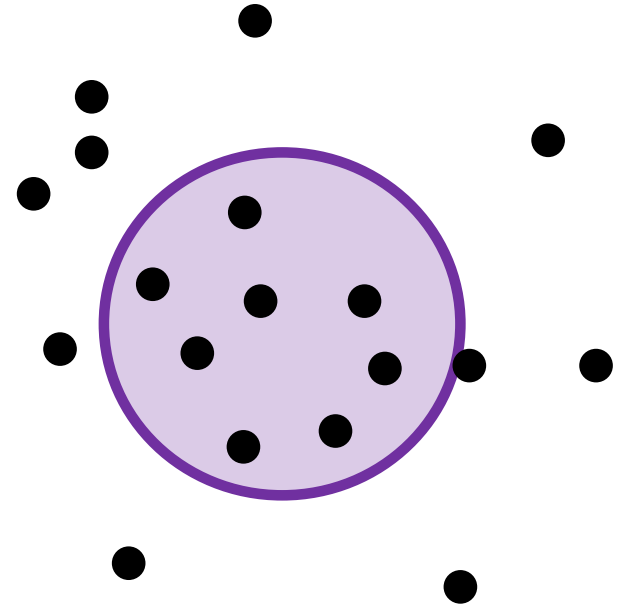


What if the data structure depended on ...the data ?

- Why is the answer not obvious ?
- It is often possible to get a data structure that works well when the data *has some structure* (clusters, low-dimensional subspace, bounded growth etc)
- The tricky part is what to do when the data *does not have* that structure

The actual idea

- **Every** point-set has some structure that can be exploited algorithmically
- Details depend on the context/problem, but at a high level:
 - Either there is dense cluster of small radius, or
 - Points are “spread”



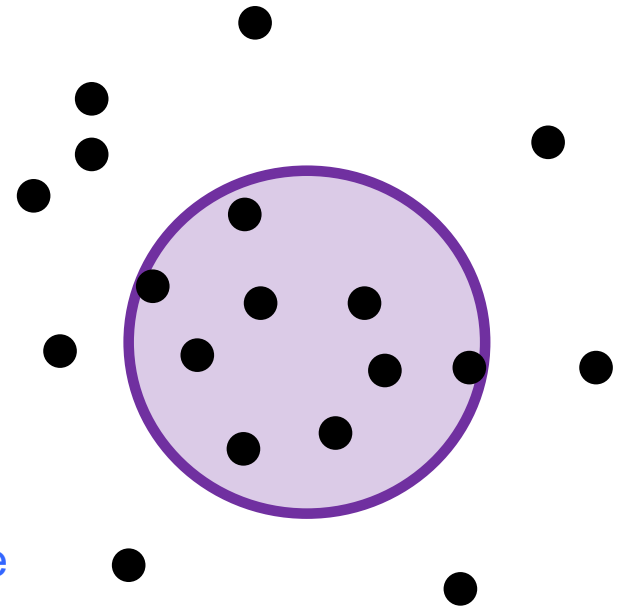
Problem 1:
Approximate *Near* Neighbor

Basic Version

P =input pointset, r =radius, c =approximation

Preprocessing:

1. As long as there is a ball B_i of radius $O(cr)$ containing T points in P
 - $P = P - B_i$
 - $i = i + 1$
2. Build the basic LSH data structure on P
No dense clusters – better performance
3. For each ball B_i build a specialized data structure for $B_i \cap P$
Diameter bounded by $O(cr)$ – better performance



Query procedure:

1. Query the main data structure
2. Query all data structures for balls that are “close” to the query

Results

- Theory (c-approximation):

Algorithm	Query Time	Index Space
Data-oblivious LSH	dn^{1/c^2}	n^{1+1/c^2}
Andoni, Indyk, Nguyen, Razenshteyn, SODA'14	$dn^{0.87/c^2 + o(1/c^3)}$	$n^{1+0.87/c^2 + o(1/c^3)}$
Andoni-Razenshteyn, STOC'15	$dn^{1/(2c^2-1)}$	$n^{1+1/(2c^2-1)}$
Andoni et al, SODA'17	tradeoff	

- Practice:

- Andoni-Indyk-Laarhoven-Razenshteyn-Schmidt (NIPS'15)
- FALCONN (Razenshteyn-Schmidt)
 - Cross-polytope LSH, plus multi-probe, fast random projections,...

ANN-Benchmarks (third party)

Info

ANN-Benchmarks is a benchmarking environment for approximate nearest neighbor algorithms search. This website contains the current benchmarking results. Please visit <http://github.com/maumueller/ann-benchmarks/> to get an overview over evaluated data sets and algorithms. Make a pull request on [Github](#) to add your own code or improvements to the benchmarking system.

Benchmarking Results

Results are split by distance measure and dataset. In the bottom, you can find an overview of an algorithm's performance on all datasets. Each dataset is annotated by ($k = \dots$), the number of nearest neighbors an algorithm was supposed to return. The plot shown depicts *Recall* (the fraction of true nearest neighbors found, on average over all queries) against *Queries per second*. Clicking on a plot reveals detailed interactive plots, including approximate recall, index size, and build time.

Machine Details

All experiments were run in Docker containers on Amazon EC2 *c4.2xlarge* instances that are equipped with Intel Xeon E5-2666v3 processors (4 cores available, 2.90 GHz, 25.6MB Cache) and 15 GB of RAM running Amazon Linux. For each parameter setting and dataset, the algorithm was given thirty minutes to build the index and answer the queries.

Raw Data & Configuration

Please find the raw experimental data [here](#) (13 GB). The query set is available [queries-sisap.tar](#) (7.5 GB) as well. The algorithms used the following parameter choices in the experiments: $k = 10$ and $k=100$.

Updates

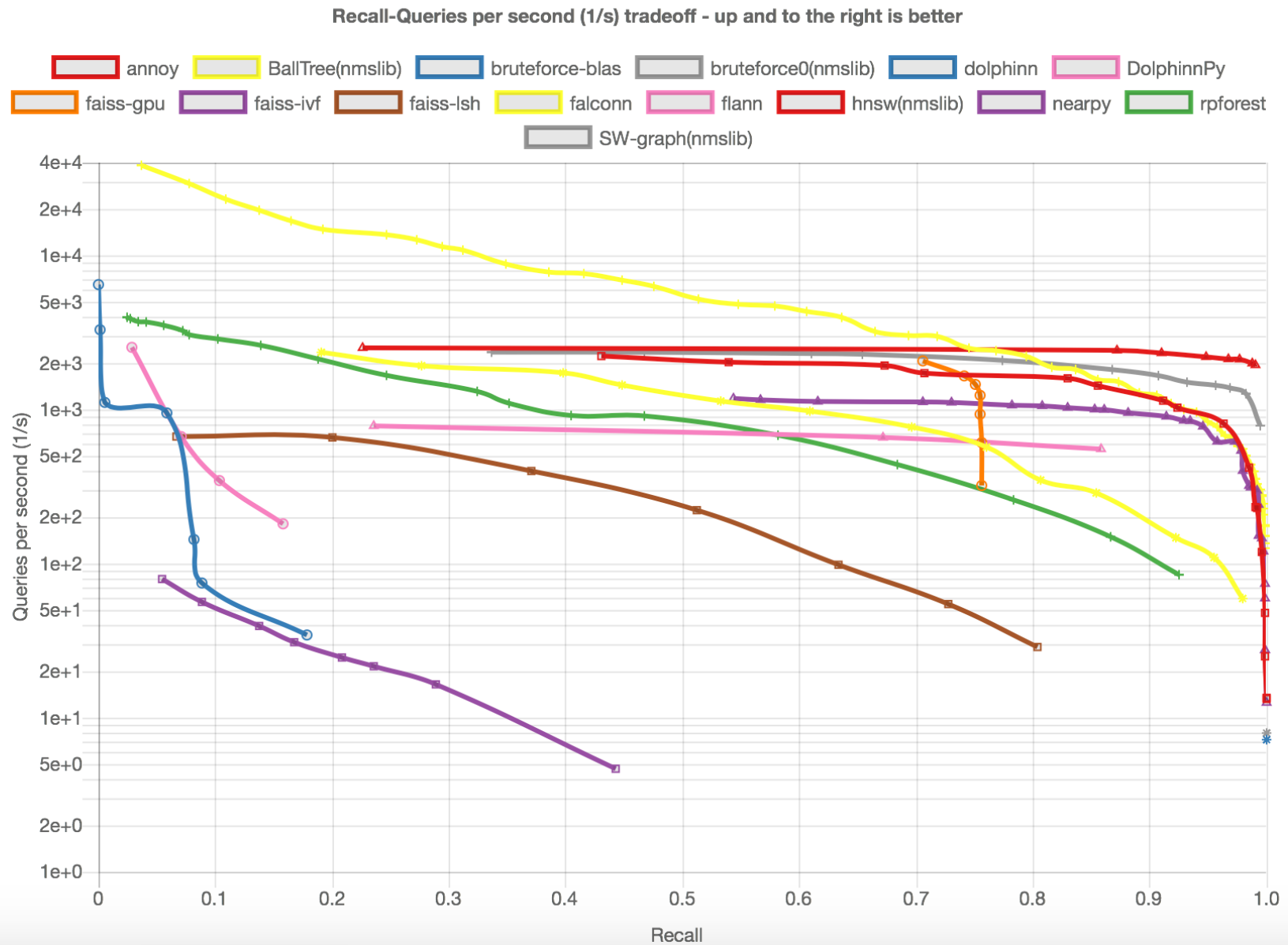
- 18-10-2017: Included FAISS-IVF

Contact

ANN-Benchmarks has been developed by Martin Aumüller (maau@itu.dk), Erik Bernhardsson (mail@erikbern.com), and Alec Fainfull (alef@itu.dk). Please use [Github](#) to submit your implementation or improvements.

Plots for glove-data (k = 10)

Recall/Queries per second (1/s)



Problem 2: Metric compression

Metric compression

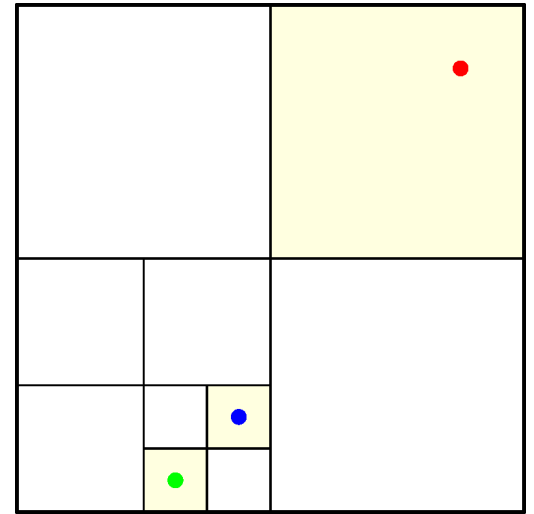
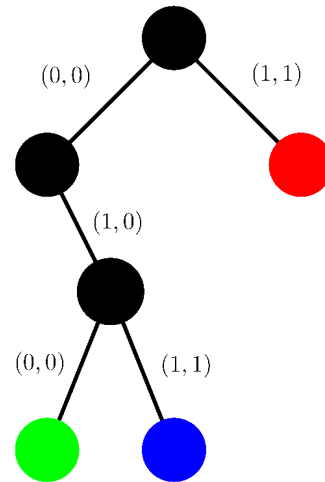
Approach	Bound	Reference	Note
No compression	$O(nd \log n)$		
Random projection	$O\left(\frac{n \log^2 n}{\varepsilon^2}\right)$	Johnson-Lindenstrauss 1984	

Intuition

- Either there is dense cluster of small radius
 - Store cluster center and displacements from it
- Or points are “spread”
 - Only few distance scales to worry about
 - $O\left(\frac{\log n}{\epsilon^2}\right)$ bits per point per scale

Quad tree

- Quad tree in R^d
- Can be seen as a binary expansion in each dimension
- Need $\Theta(\log n)$ levels
- $\Theta(nd \log n) = \Theta\left(\frac{n \log^2 n}{\varepsilon^2}\right)$ bits total

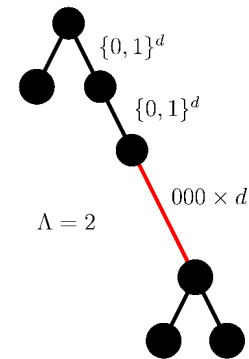


Pruning

- Compress paths of degree-2 nodes of length $> \Lambda$
- Leave only the first Λ edges
- Decompress the remainder into zeros
- The total size is

$$O(nd\Lambda) = O\left(\frac{n\Lambda \cdot \log n}{\varepsilon^2}\right)$$

- Store the Eulerian tour of the tree
- Set $\Lambda = O(\log \log n + \log(1/\varepsilon))$

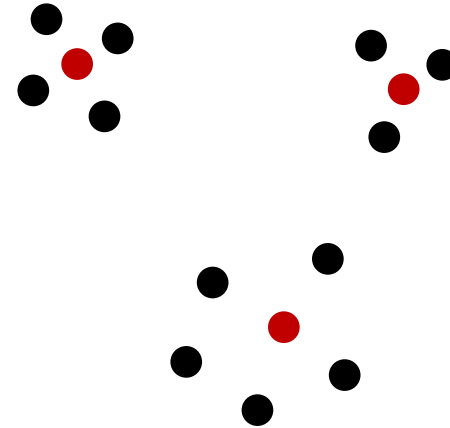


Experiments: baselines

- We compare the new algorithm (QuadSketch) with two baselines:
 - Grid-based compression (round every coordinate)
 - Product Quantization (PQ) [**Jegou, Douze, Schmid 2009**]
 - Good in practice (and popular)
 - No theoretical guarantees (for arbitrary data sets)

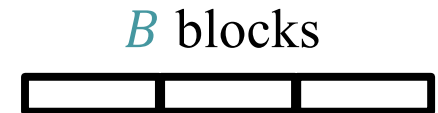
PQ: k -means

- A way to cluster a dataset
- Find k centers $c_1, c_2, \dots, c_k$
- Minimize $\sum_i \|p_i - c_{u_i}\|_2^2$,
where
 - c_{u_i} is the center closest to p_i
- NP-hard, good heuristics
- Can be used as a sketch ($\log k$ bits)
- Estimate: $\|c_q - c_s\|_2$
- Problem: **too many centers**



PQ: blocks

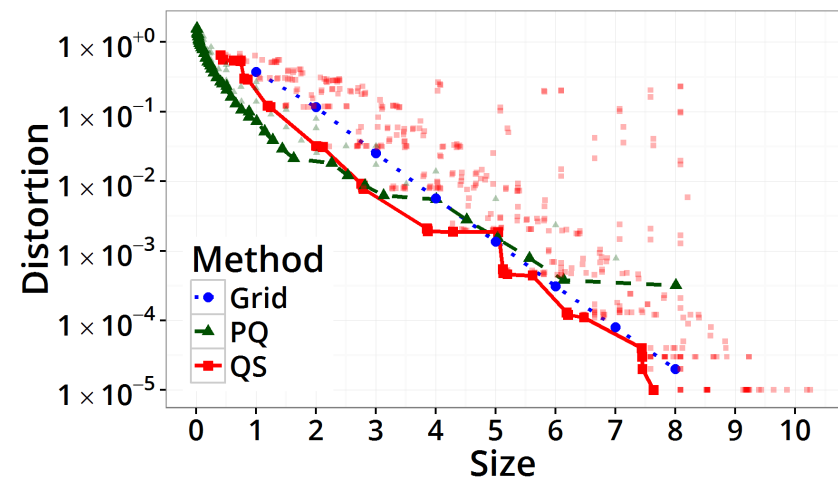
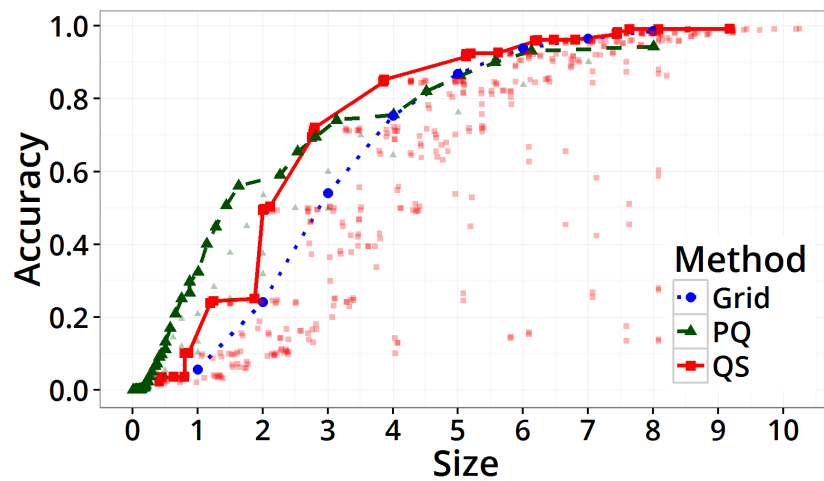
- Introduced in Jegou-Douze-Schmid'09
- Partition a vector in B blocks
- Project a dataset on each block, and find k -means in each
- Sketch of $B \log k$ bits (k^B landmarks as opposed to k)
- Estimation:
 - One-time cost $O(dk)$
 - Plus $O(B)$ per point
- We add the same trick to Quads sketch



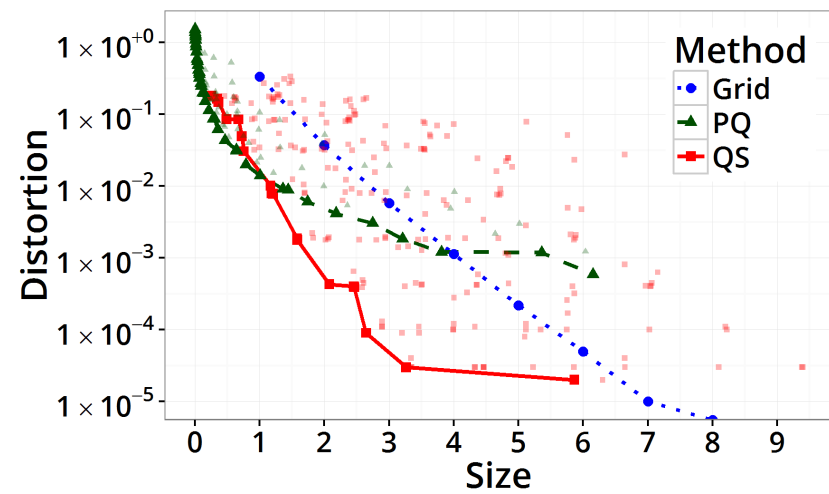
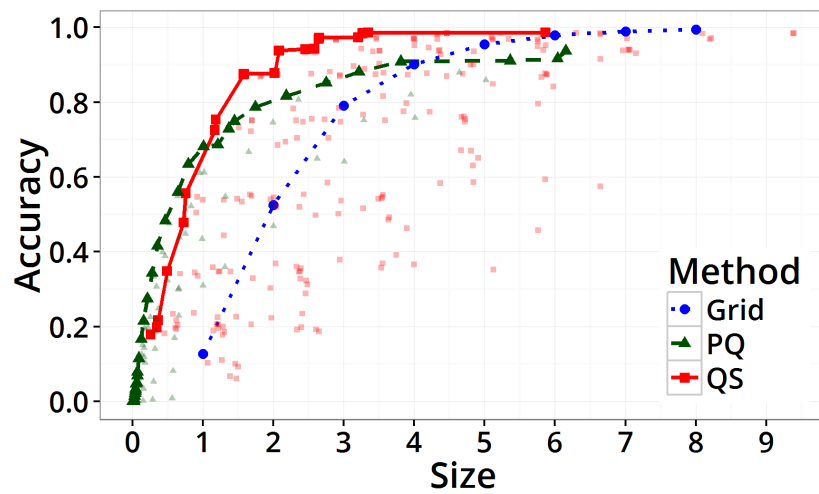
Experiments: protocol and datasets

- We look at:
 - 1-NN accuracy
 - Distortion of 1-NN
- Datasets:
 - **SIFT1M** (1M points, 128 dimensions)
 - **MNIST** (60K points, 784 dimensions)
 - **Taxi** (9K points, 48 dimensions)
 - **Diagonal** (synthetic, 10K points, 128 dimensions)

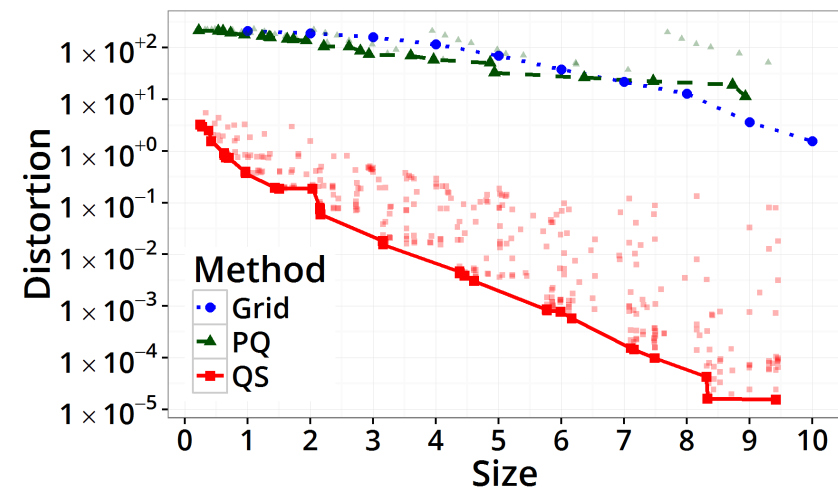
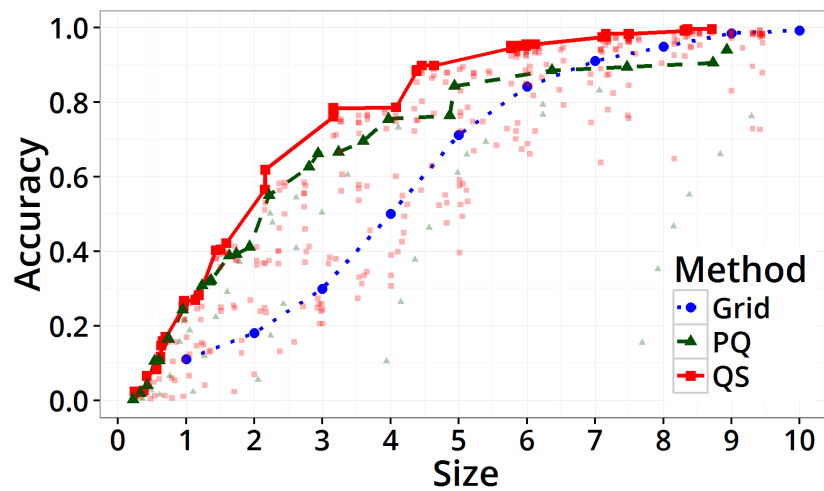
Results: SIFT1M



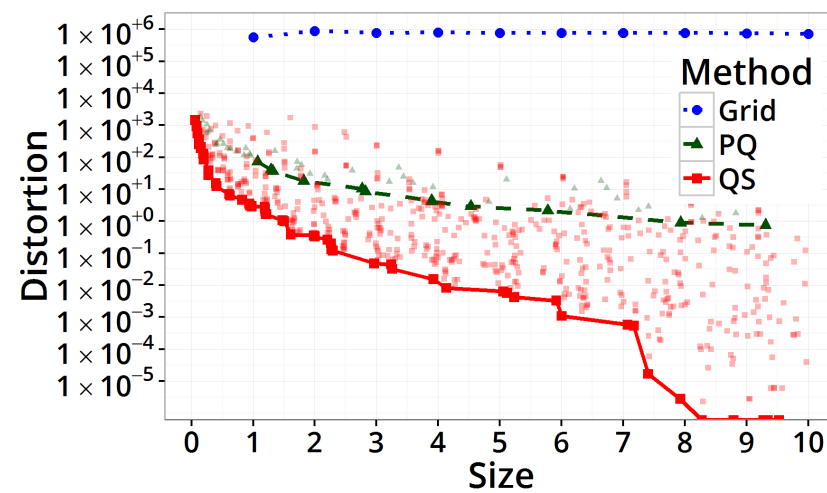
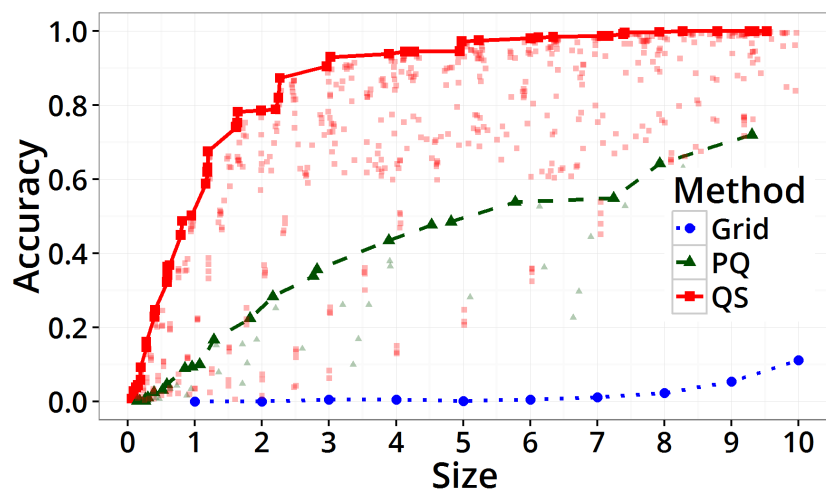
Results: MNIST



Results: Taxi



Results: Diagonal



Conclusions

- Data helps designing data structures
- ..provably, for general high-dimensional pointsets
- Code:
 - FALCONN: Google it
 - Quads sketch: Google it