
David C. Anastasiu$^{1,2}$, Huzefa Rangwala$^3$, and Andrea Tagarelli$^4$

$^1$Computer Engineering, San Jose State University, CA
$^1$Computer Science & Engineering, Santa Clara University, CA
$^2$Computer Science & Engineering, George Mason University, VA
$^3$DIMES, University of Calabria, Italy

25th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, August 2019
Part III: Approximate Search

David C. Anastasiu, San José State University [ david.anastasiu@sjsu.edu ]

Starting September:
Department of Computer Science and Engineering
Santa Clara University
Tutorial Outline

- **Part I: Problems and Data Types**
  - Dense, sparse, and asymmetric data
  - Bounded nearest neighbor search
  - Nearest neighbor graph construction
  - Classical approaches and limitations

- **Part II: Neighbors in Genomics, Proteomics, and Bioinformatics**
  - Mass spectrometry search
  - Microbiome analysis

- **Part III: Approximate Search**
  - Locality sensitive hashing variants
  - Permutation and graph-based search
  - Maximum inner product search

- **Part IV: Neighbors in Advertising and Recommender Systems**
  - Collaborative filtering at scale
  - Learning models based on the neighborhood structure

- **Part V: Filtering-Based Search**
  - Massive search space pruning by partial indexing
  - Effective proximity bounds and when they are most useful

- **Part VI: Neighbors in Learning and Mining Problems in Graph Data**
  - Neighborhood as cluster in a complex network system
  - Neighborhood as influence trigger set
Approximate near(est) neighbor search

• Input: $P, q, r, \epsilon$

• Output:
  • If some points exists with $\text{dist}(p, q) \leq r$, output any such ANN and return YES
  • If no points exists with $\text{dist}(p, q) \leq (1 + \epsilon)r$, return NO
  • Otherwise, return any point with $\text{dist}(p, q) \leq (1 + \epsilon)r$
Directions of approximate methods

• Plethora of methods, especially in recent years
• Deep learning and abundance of data lead to a push towards
  • Faster and faster methods
  • Methods that can search 1 Billion+ objects
  • While accuracy is sought after, scaling and efficiency seem to be of primary concern
• Methods fall into one or more of the following categories:
  • Locality-sensitive hashing
  • Graph traversal
  • Quantization
  • GPU-focused
  • Other
Locality sensitive hashing

• A set of functions is *locality-sensitive*, if, for a random hash function \( h \) in the set, for any pair of points \( p, q \):
  • \( \Pr[h(p) = h(q)] \) is “high” if \( p \) is “close” to \( q \)
  • \( \Pr[h(p) = h(q)] \) is “low” if \( p \) is “far” from \( q \)
Locality sensitive hashing

- **Preprocessing**:  
  - Hash the data-point using several LSH functions so that probability of collision is higher for closer objects (increases recall, but hurts precision)  
    - E.g., partition with random Gaussian noise vectors for Euclidean distance search

- **Querying**:  
  - Hash query point and retrieve elements in the buckets containing the query point

- **Give-and-pull between # hashes and # tables**  
  - Increase precision by repeating the search multiple times (use several hash functions) and keeping only common points (set intersection)  
  - Increase recall by repeating the search multiple times (use several hash functions) and keeping all found points (set union)
Recent directions in LSH

• C2LSH
  • Identifies candidates by collision counting
  • Good candidates express a large number of collisions with the query when tested against dynamically-generated compound hash functions derived from a set of $m$ initial functions.

• QALSH
  • A query-aware data-dependent LSH scheme
  • Boundaries of the buckets are query-dependent, decided once query is received
    • Leads to improved effectiveness
Quantization methods

• Main idea is to reduce the size of the vectors so more can fit in memory/cache
  • A quantizer is a function mapping real-valued vectors into a vector of integers from a finite set

• PQ
  • Decomposes the space into a Cartesian product of low dimensional subspaces, which are separately quantized.
  • A vector is represented by a short code composed of its subspace quantization indices.

• OPQ
  • Improves quantization accuracy by a linear transformation applied to the input vectors
Graph traversal method

• Main idea is to pre-process the objects and build a $k$-NNG including all database objects

• At search time, the $k$-NNG is traversed in search of better and better neighbors for the query

• The methods largely differ in:
  • Starting point in traversing the graph
    • NGT, e.g., uses a secondary tree structure to identify better/optimum starting points
  • Traversal strategy
    • kGraph, e.g., looks only among the neighbors of the query’s current neighbors for possible better neighbors
  • Local or global graph construction
    • HNSW creates a tree of $k$-NNGs at different granularities
      • During the search, it also expands the search space with less-similar candidates, as a way to prevent getting stuck in a local structure (think simulated annealing, or restarts in PageRank).
So, what’s the best search method?

• It depends on your data (primarily dimensionality and number of objects)

• Erik Bernhardsson put together a benchmarking framework for approximate nearest neighbor codes

• It provides
  • Realistic data sets
  • Standard evaluation metrics
  • A platform to showcase algorithm comparisons

• It requires
  • Python bindings for the tested methods

• https://github.com/erikbern/ann-benchmarks
As of today...

---

**glove-100-angular**

Recall-Queries per second (1/s) tradeoff - up and to the right is better

**gist-960-euclidean**

Recall-Queries per second (1/s) tradeoff - up and to the right is better

https://github.com/erikbern/ann-benchmarks
References

Stochastic/Approximate methods:

Locality-Sensitive Hashing:


Quantization:


GPU:


References

*Graph traversal:*


*Mixed/Other:*


[ANNOY] https://github.com/spotify/annoy
