Towards a Model for LSH

Li Wang

Wufong University, China

Abstract—As data volumes continue to grow, clustering and outlier detection algorithms are becoming increasingly time-consuming. Classical index structures for neighbor search are no longer sustainable due to the "curse of dimensionality". Instead, approximated index structures offer a good opportunity to significantly accelerate the neighbor search for clustering and outlier detection and to have the lowest possible error rate in the results of the algorithms. Locality-sensitive hashing is one of those. We indicate directions to model the properties of LSH.

Index Terms—LSH.

I. BASIC DEFINITION

The idea is to consider the convex, Voronoi-like cells of k-means as a d-dimensional grid of hyper-cubic cells which have a volume such that an expectation of \( n/k \) points are located within each. We denote the side length of these cells with \( b \) The query is modeled as a ball with perimeter \( s \), centered at origin. In the beginning, we assume that the ball is actually from a maximum metric such that the query is as well a hypercube with side length \( s \).

In a locality sensitive hashing we use a number \( m \) of alternative clusterings, and we refine for each of the \( m \) clusterings exactly one cell, in which the query point (origin) is located. Therefore, the selectivity of the query response corresponds to the intersection of the union of all \( m \) cells with the query ball. We start with the analysis of the case \( s = b = 1 \). We assume that each of the \( m \) cells is selected uniformly and independently such that the origin (query point) is located inside, so the upper boundary of each cell in each dimension is uniformly taken from the interval \([0, \frac{1}{2}]\) and the lower boundary is consequently from \([-\frac{1}{2}, 0]\).

In the following we will use several definite integrals which we denote in a bit unusual way which helps for clarity we will write

\[
\int \ldots (0 \leq x \leq 1) \text{ instead of } \int_0^1 \ldots \text{ } dx.
\]

We will need a few integrals throughout this paper:

\[
\begin{align*}
\int x + \frac{1}{2} \ (0 \leq x \leq \frac{1}{2}) & = \frac{3}{8} \\
\int (x + \frac{1}{2})(y + \frac{1}{2}) \ (0 \leq x, y \leq \frac{1}{2}) & = \frac{9}{16} \\
\int \ldots (x_1 + \frac{1}{2}) \ldots (x_d + \frac{1}{2}) \ (0 \leq x_1, \ldots, x_d \leq \frac{1}{2}) & = \left(\frac{3}{8}\right)^d \\
\int \min(x_1, x_2) + \frac{1}{2} \ (0 \leq x_1, x_2 \leq \frac{1}{2}) & = \frac{5}{24}
\end{align*}
\]

\[
\begin{align*}
\int \ldots \int \min(x_1, \ldots, x_d) + \frac{1}{2} \ (0 \leq x_1, \ldots, x_d \leq \frac{1}{2}) & = \\
& = d \cdot \int x^{d-1} \cdot (x + \frac{1}{2}) \ (0 \leq x \leq \frac{1}{2}) = \frac{2d + 1}{(d + 1) \cdot 2^{d+1}} \\
\int y - v \ (0 \leq y \leq \frac{1}{2}, -\frac{1}{2} \leq v \leq 0) & = \frac{1}{8} \\
\int \ldots \int (y_1 - v_1) \ldots (y_d - v_d) \ (0 \leq y_1, \ldots, y_d \leq \frac{1}{2}, -\frac{1}{2} \leq v_1, \ldots, v_d \leq 0) & = \\
& = \frac{2m + 1}{8d \cdot (m + 1) \cdot 2^{m+1}}.
\end{align*}
\]

Definition 1. \( p(m, \ell, d) \) is the (hyper-) volume of the hypercube representing the query which is occupied by at least \( \ell \) of the \( m \) cells (with \( \ell \leq m \)). Here the \( m \) cells are uniformly selected in \( \mathbb{R}^d \) such that the query point is inside the cell.

Note that \( p(m, 1, d) \) corresponds to the selectivity of the hashing provided that \( b \) and \( s \) are of equal size.

For the case \( m = \ell = 1 \) we can easily derive the closed formula of \( p(1, 1, d) \). We simply have to form, in each quadrant, the (identical) expectation with which the query is occupied by the cell:

\[
n(1, 1, d) = 2^d \int \ldots \int (x_1 + \frac{1}{2}) \ldots (x_d + \frac{1}{2}) \ (0 \leq x_1, \ldots, x_d \leq \frac{1}{2}) = \left(\frac{3}{4}\right)^d
\]

The case \( m > 1, \ell = 1 \) can be reduced to the case \( m = 1 \):

\[
p(m, 1, d) = \binom{m}{1} \cdot p(1, 1, d) - \binom{m}{2} \cdot p(1, 2, d) + \binom{m}{3} \cdot p(1, 3, d) - \ldots \pm \binom{m}{i} \cdot p(1, i, d)
\]

For the case \( m = 1, \ell = 2 \), we start with the analysis of \( d = 1 \) and construct this case with two variables \( x \) and \( y \). We
want to estimate the area which is covered by \( x \) and \( y \), which is

\[
p(1, 2, 1) = \iint \left\{ \begin{array}{ll}
1 - |x - y| & \text{if } x < 0 \text{ xor } y < 0 \\
1 - \max(|x|, |y|) & \text{otherwise}
\end{array} \right.
\]

This is an independent information in each dimension such that

\[
p(1, 2, d) = p(1, 2, 1)^d = \left( \frac{7}{12} \right)^d.
\]

For a generalization to \( \ell = 3 \) we make a similar case distinction:

\[
p(1, 3, 1) = \iint \left\{ \begin{array}{ll}
1 - \max(x, y, z) & \text{if } x, y, z \geq 0 \\
1 - \max(x, y) + z & \text{if } x, y \geq 0 \text{ and } z < 0 \\
\ldots
\end{array} \right.
\]

where the first case occurs in 2 octants and the second case in the remaining 6 octants. We solve the first case by letting

\[
2 \cdot \iiint 1 - \max(x, y, z) = 6 \cdot \int (1 - x) \cdot x^2 = \frac{5}{12}
\]

and similarly the second case

\[
6 \cdot \iiint 1 - \max(x, y) + z = 12 \cdot \int (1 - x + z) \cdot x = \frac{5}{18}.
\]

For the general case, we consider \( \ell \) variables \( x_1, \ldots, x_\ell \). Again we consider a solution space where we distinguish if each variable is greater or less than 0. All variables are equivalent, thus we consider all quadrants of the solutions space equally where the same number of variables is \( \leq 0 \). There is a number of

\[
\binom{\ell}{i} \quad \text{quadrants of the solution space having } i \text{ out of } \ell \text{ variables } \leq 0.
\]

Therefore, we have

\[
p(1, \ell, 1) = \sum_{0 \leq i \leq \ell} \binom{\ell}{i} \cdot \iiint 1 - \max(x_1, \ldots, x_i) + \min(x_{i+1}, \ldots, x_\ell) \geq 0
\]

II. RELATED WORK

There are various approaches for different data types. The data can be of any type, as long as a distance function exists. Fixed-length text data often uses Hamming distance [1] and the similarity between variable length text is often measured using the edit distance [2]. A common measure for set data is the Jaccard distance [3], [4], whereas the similarity of documents is processed with cosine-like similarity measures [5], [6].

Approximate nearest neighbor search techniques can also be applied to the similarity join problem, however without guarantees on completeness and exactness of the result. There may be false positives as well as false negatives. Recently an approach [7] to Locality Sensitive Hashing (LSH) is used on a representative point sample, to reduce the number of lookup operations. LSH is of interest in theoretical foundational work, where a recursive and cache-oblivious LSH approach [8] was proposed. The topic of approximate solutions for the similarity join is also an emerging field in deep learning [9]. There are approximate approaches which target low dimensional cases (spatial joins in 2–3 dimensions [10]) or higher (10–20) dimensional cases [11]. Very high-dimensional cases, with dimensions of 128 and above have been targeted with Symbolic Aggregate approxImation (SAX) techniques [12]) to generate approximate candidates. SAX techniques rely on several indirect parameters like PAA size or the ISAX alphabet size.

There are preconstructed indexing techniques, which are based on space-filling curves and applied to the similarity join problem. Specifically, where the data is sorted efficiently with respect to one or more Z-order curves [13]–[15] in order to test the intersection of the hypercubes in the datastructures. Others propose space-filling curves, to reduce the storage cost for the index [16]. LESS [13] targets GPUs and not multi-core environments. ZC and MSJ [14] as well as the SPB-tree index [16], although simple, they require space transformations and preprocessing, which make them hard to parallelize.

EGO family of \( \epsilon \)-join algorithms. The EGO-join algorithm is the first algorithm in this family introduced by Böhm et al. in [17]. The Epsilon Grid Order (EGO) was introduced as a strict order (i.e. an order which is irreflexive, asymmetric and transitive). It was shown that all join partners of some point \( x \) lie within an \( \epsilon \)-interval, of the Epsilon Grid Order. Algorithms of the EGO family exploit this knowledge for the join operation. The EGO-join has been re-implemented as a recursive variant with additional heuristics, to quickly decide whether two sequences are non-join-able [18]. Further improvements proposed two new members of this family, the EGO* [19] algorithm and its extended version called Super-EGO [20] target multi-core environments using a multi-process/multi-thread programming model. Super-EGO proposes a dimensional reordering [20]. In the experiments Super-EGO encounters some difficulties with uniformly distributed data, particularly when the number of data objects exceeds millions of points or the dimensionality is above 32.
If the similarity join runs multiple times on the same instances of the data, one might consider **index-based approaches** \([16, 21, 22]\), such as R-tree \([23]\) or M-tree \([24]\). Index-based approaches have the potential to reduce the execution time, since the index stores pre-computed information that significantly reduces query execution time. This pre-computational step could be costly, especially in the case of List of Twin Clusters (LTC) \([22]\), where the algorithm needs to build joint or combined indices for every pair of points in the dataset. The D-Index \([25]\) and its extensions (i.e. eD-Index \([26]\) or i-Sim index \([27]\)) build a hierarchical structure of index levels, where each level is organized into separable buckets and an exclusion set. The most important drawback of D-Index, eD-Index and i-Sim is that they may require rebuilding the index structure for different \(\epsilon\).

**Data partitioning across multiple machines** is not the main focus of this paper where we assume that the data fits into main memory. The case of relational join algorithms has been studied extensively in the past \([28, 30]\). The similarity join has been successfully applied in the distributed environment with different MapReduce variants \([30, 32]\). Another distributed version is proposed in \([33]\). There, a multi-node solution with load-balancing is used, that does not require re-partitioning on the input data. This variant focuses on minimizing data transfer, network congestion and load-balancing across multiple nodes.

The similarity join has been already implemented for **Graphics Processing Units (GPUs)**. In \([34]\), the authors use a directory structure to generate candidate points. On datasets with 8 million points, the proposed GPU algorithm is faster than its CPU variant, when the \(\epsilon\)-region has at least 1 or 2 average neighbors. **LSS** \([15]\) is another similarity join variant for the GPU, which is suited for high dimensional data. Unfortunately both \([15]\) and \([34]\) are targeted to NVIDIA GPUs and have been optimized for an older version of CUDA.

### A. Cache-oblivious Algorithms

Cache-oblivious algorithms \([35]\) have attracted considerable attention as they are portable to almost all environments and architectures. Algorithms and data structures for basic tasks like sorting, searching, or query processing \([36]\) and for specialized tasks like ray reordering \([37]\) or homology search in bioinformatics \([38]\) have been proposed. Two important algorithmic concepts of cache-oblivious algorithms are localized memory access and divide-and-conquer. The Hilbert curve integrates both ideas. The Hilbert curve defines a 1D ordering of the points of an 2-dimensional space such that each point is visited once. Bader et al. proposed to use the Peano curve for matrix multiplication and LU-decomposition \([39], 40]\). The algorithms process input matrices in a block-wise and recursive fashion where the Peano curve guides the processing order and thus the memory access pattern. In \([41]\), cache-oblivious loops have been applied to K-means clustering and matrix multiplication.

### B. Optimized Techniques for Specific Tasks or Hardware

The library BLAS (Basic Linear Algebra Subprograms) \([42]\) provides basic linear algebra operations together with programming interfaces to C and Fortran. BLAS is highly hardware optimized: specific implementations for various infrastructures are available, e.g. ACML for AMD Opteron processors or CUBLAS for NVIDIA GPUs. The Math Kernel Library (MKL) contains highly vectorized math processing routines for Intel processors. These implementations are very hardware-specific and mostly vendor-optimized. Moreover, they are designed to efficiently support specific linear algebra operations. Experiments demonstrate that the cache-oblivious approach reaches a performance better than BLAS on the task of the similarity join for points of dimensions in the range of \([2, ..., 64]\).

### III. Conclusion

This survey reviews several approaches of HPDM from many research groups world wide. Modern computer hardware supports the development of high-performance applications for data analysis on many different levels. The focus is on modern multi-core processors built into today’s commodity computers, which are typically found at university institutes both as small server and workstation computers. So they are deliberately not high-performance computers. Modern multi-core processors consist of several (2 to over 100) computer cores, which work independently of each other according to the principle of “multiple instruction multiple data” (MIMD). They have a common main memory (shared memory). Each of these computer cores has several (2-16) arithmetic-logic units, which can simultaneously carry out the same arithmetic operation on several data in a vector-like manner (single instruction multiple data, SIMD). HPDM algorithms must use both types of parallelism (SIMD and MIMD), with access to the main memory (centralized component) being the main barrier to increased efficiency.

### REFERENCES

[1] ThienLuan Ho, Seungrok Oh, and Hyunjin Kim. New algorithms for fixed-length approximate string matching and approximate circular string matching under the hamming distance. The Journal of Supercomputing, 74(5):1815–1834, 2018.

[2] Chuan Xiao, Wei Wang, and Xuemin Lin. Ed-join: an efficient algorithm for similarity joins with edit distance constraints. PVLDB, 1(1):933–944, 2008.

[3] Dong Deng, Yufei Tao, and Guoliang Li. Overlap set similarity joins with theoretical guarantees. In SIGMOD Conf. 2018, pages 905–920, 2018.

[4] Chuan Xiao, Wei Wang, Xuemin Lin, Jeffrey Xu Yu, and Guoren Wang. Efficient similarity joins for near-duplicate detection. ACM Trans. Database Syst., 36(3):15:1–15:41, 2011.

[5] Nikolaus Augsten and Michael H. Böhlen. Similarity Joins in Relational Database Systems. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2013.

[6] Zeyuan Shang, Xayiao Liu, Guoliang Li, and Jianhua Feng. K-join: Knowledge-aware similarity join. In ICDE, pages 23–24, 2017.

[7] Chenyun Yu, Sarana Nutanong, Hangyu Li, Cong Wang, and Xingliang Yuan. A generic method for accelerating lsh-based similarity join processing. IEEE Trans. Knowl. Data Eng., 29(4):712–726, 2017.

[8] Rasmus Pagh, Ninh Pham, Francesco Silvestri, and Morten Stöckel. Ifficient similarity join. Algorithmica, 78(4):1263–1283, 2017.
[81] Wei Ye, Samuel Maurus, Nina Hubig, and Claudia Plant. Generalized independent subspace clustering. In Francesco Bonchi, Josep Domingo-Ferrer, Ricardo Baeza-Yates, Zhi-Hua Zhou, and Xindong Wu, editors, IEEE 16th International Conference on Data Mining, ICDM 2016, December 12-15, 2016, Barcelona, Spain, pages 569–578. IEEE Computer Society, 2016.

[82] Claudia Plant. Dependency clustering across measurement scales. In Qiang Yang, Deepak Agarwal, and Jian Pei, editors, The 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD ’12, Beijing, China, August 12-16, 2012, pages 361–369. ACM, 2012.

[83] Christian Böhm and Florian Krebs. The k-nearest neighbour join: Turbo charging the KDD process. Knowl. Inf. Syst., 6(6):728–749, 2004.

[84] Theodore Bially. Space-filling curves: Their generation and their application to bandwidth reduction. IEEE Trans. Information Theory, 15:658–664, 1969.

[85] P Prusinkiewicz. Graphical applications of l-systems. In Proceedings on Graphics Interface ’86/Vision Interface ’86, pages 247–253, 1986.