DOI 10.1007/s10559-018-0016-1
Cybernetics and Systems Analysis, Vol. 54, No. 1, January, 2018

NEW MEANS OF CYBERNETICS, INFORMATICS, COMPUTER ENGINEERING, AND SYSTEMS ANALYSIS

INDEX STRUCTURES FOR FAST SIMILARITY SEARCH FOR REAL-VALUED VECTORS. I

D. A. Rachkovskij

UDC 004.22 + 004.93′11

Abstract. *This survey paper considers index structures for fast similarity search for objects represented by real-valued vectors. Index structures based on locality-sensitive hashing and their modifications are discussed. The ideas of specific algorithms, including the recently proposed ones, are stated. Their interrelations and some theoretical aspects are discussed.*

Keywords: *similarity search, nearest neighbor, near neighbor, index structure, locality-sensitive hashing, locality-sensitive filtering.*

INTRODUCTION

Similarity search is a variant of information search in which objects from some base (set) of objects relevant to a query object are determined based on values of some measures (functions) of similarity or dissimilarity (distance) between representations of objects.

We consider that the object representations and distance/similarity measure underlain a search are given. Similarity search is most simply implemented by computing the similarity values of a query object and all N base objects and returning the objects meeting the query conditions. This "linear" (or "sequential") search frequently turns out to be inadmissibly slow, especially for large bases, complicated multicomponent object representations, and computationally intensive similarity measures.

An approach to accelerating linear similarity search is to quickly (though approximately) estimate values of all distances/similarities (see [1, 2]). Another approach consists of constructing a data structure (index structure, IS) constructed from the base and such that its use in implementing a similarity search query would allow one to reduce the number of computations of the similarity between the query object and other objects in comparison with linear search (sublinear search with respect to N). Note that algorithms of both approaches often accelerate search at the expense of obtaining results that do not completely coincide with the results of (exact) linear similarity search, i.e., provide an approximate similarity search.

A review of ISs that are developed within the second approach and do not require explicit access to object representations except for computing distances/similarities is presented in [3]. In [4], ISs are considered for similarity search for objects that are binary vectors. The present article considers main types of ISs for fast similarity search for objects represented by real-valued vectors (their components are real numbers). Such vectors are widely used, for example, for representing texts, images, etc. [5–9]. Though ISs from [3] can be used for vectors, vectors provide more tools for indexing owing to the possibilities of accessing their components and their subsets, explicitly specifying boundaries of domains of a vector space, applying specialized operations and algorithms (averaging and clustering), etc.

This survey consists of two parts, and this paper is its first part. The ISs are mainly considered that are practically implemented or that have a perspective of such an implementation and are constructed without information from a teacher. In this part of the survey paper, ISs are discussed that are destined for approximate similarity search based on hashing preserving similarity (Sec. 2). An introduction to the problematic of similarity search is presented to Sec. 1.

International Scientific-Educational Center of Information Technologies and Systems, NAS of Ukraine and MES of Ukraine, Kyiv, Ukraine, *dar@infrm.kiev.ua*. Translated from Kibernetika i Sistemnyi Analiz, No. 1, January–February, 2018, pp. 168–183. Original article submitted August 23, 2017.

1. BASIC CONCEPTS

For distances and similarities used in ISs, we will consider the definitions of types of similarity search queries, problems of exact search, and the need for the transition to approximate search to fast execute queries and also the principles of construction and application of such ISs.

1.1. Distances and similarities. To estimate similarities between objects, distance, i.e., "dissimilarity," is used. To large similarity values correspond small distance values. Many distances are metrics, i.e., submit to metric axioms such as the triangle inequality, etc. For real-valued vectors $\mathbf{a}$ and $\mathbf{b}$ of dimension D, Minkowski distances L_s of different orders s: $||\mathbf{a}-\mathbf{b}||_s = \left(\sum_{i=1}^{D} |a_i - b_i|^s\right)^{1/s}$ are widely used. The metric distances L_2 (Euclidean $||\mathbf{a}-\mathbf{b}||_2$), L_1 (Manhattan $||\mathbf{a}-\mathbf{b}||_1$), and L_∞ (Chebyshev distance $||\mathbf{a}-\mathbf{b}||_\infty$) are most widespread. When $0 < s < 1$, fractional distances are obtained that are not metrics.

For similarities, large values correspond to more similar objects. One of similarity measures for vectors is the scalar product $\text{sim}_{\text{dot}}(\mathbf{a}, \mathbf{b}) \equiv \langle \mathbf{a}, \mathbf{b} \rangle = \sum_{i=1}^{D} a_i b_i$. Note that $||\mathbf{a}-\mathbf{b}||_2^2 = ||\mathbf{a}||_2^2 + ||\mathbf{b}||_2^2 - 2\langle \mathbf{a}, \mathbf{b} \rangle$ and also that the value of $\text{sim}_{\text{dot}}(\mathbf{a}, \mathbf{a})$ is not maximally possible. The cosine of the angle between vectors is defined as $\cos(\mathbf{a}, \mathbf{b}) = \langle \mathbf{a}, \mathbf{b} \rangle / (||\mathbf{a}||_2 ||\mathbf{b}||_2) \equiv \text{sim}_{\cos}$. The angle between $\mathbf{a}$ and $\mathbf{b}$ determines the distance $\text{dist}_{\text{ang}} = \arccos(\mathbf{a}, \mathbf{b})$. The vectors of the Euclidean unit norm ($||\mathbf{a}||_2 = 1$) are also called the vectors of the (Euclidean) unit sphere, which is denoted by S^{D-1}. The ordering of such vectors in ascending order of the Euclidean distance (or angle) is equivalent to their ordering in descending order of the scalar product or the cosine of the angle. For binary vectors, the Hamming distance $\text{dist}_{\text{Ham}}(\mathbf{a}, \mathbf{b}) = \sum_{i=1}^{D} \{a_i \neq b_i\}$ is widely used, where $\{\cdot\}$ is an indicator function. The complexity (time) of computation of the mentioned distances/similarities amounts to $O(D)$.

1.2. Exact and approximate similarity search and query types.

1.2.1. Exact similarity search queries. The base objects that represent the answer to a concrete similarity search query are determined by the type of a query and its parameters (specified by its query object, distance/similarity measure, etc.). Query objects do not necessarily belong to the base. In the present review, objects are real-valued vectors.

A range query is denoted by rNN. It returns base objects whose distances from the query object (according to the distance measure specified in the query) do not exceed the query radius r. For some r, the result of a range query can be the empty set of objects or all the objects of a concrete base. In the latter case, the acceleration of executing an rNN query in comparison with linear search is impossible in principle. Any object that is an answer to a range query is called an r-near neighbor and is denoted by rNN1. An rNN1 query returns such an object if it is present in the base.

A query of k nearest neighbors (a kNN query) returns k base objects nearest to the query object. A query of one nearest neighbor and also the object that is the nearest neighbor is denoted by NN.

rNN and kNN queries with their definitions given above are queries of exact similarity search. The definitions of types of queries in terms of similarity rather than distance are similar. Other types of queries of exact similarity search are not considered in this survey.

1.2.2. Problems of exact search and approximate similarity search. The execution time of similarity search queries using linear search in a base consisting of N objects-vectors for distance/similarity measures with computation time $O(D)$ amounts to $O(ND)$.

For data of dimension $D = 1$ (each object is represented by one number), it is easy to fast execute exact similarity search queries. N numbers representing N base objects are sorted and stored. An NN query is executed by binary search. For such an IS, memory cost amounts to $O(N)$ and search time amounts to $O(\log N)$ in the worst case [3].

A similar IS for $D \geq 2$ constructs a Voronoi diagram, i.e., divides spaces into N domains each of which corresponds to a base object and consists of points that are nearer to it than to the other base objects. Unfortunately, memory cost for storing a Voronoi diagram amounts to $O(N^{\lceil D/2 \rceil})$ [10]. The best well-known algorithm of this class for NN search, although it provides the execution time of a query $O(D^{O(1)} \log N)$ [11, 12], requires $N^{O(D)}$ memory, which is unrealistic for bases of moderate size N even with small D. For algorithms with memory cost close to linear, the time for a random query object amounts (at the average) to $\min(2^{O(D)}, DN)$ [12]. Therefore, for bases with attainable N, exact search degenerates into linear even for not too large D, which is also confirmed in practice [13].

The analysis of all well-known algorithms of exact execution of an NN query with a subliner time with respect to N shows that time or memory cost grows exponentially with respect to D [12]. According to the curse of dimensionality [12], this dependence is inevitable in exact similarity search for worst-case data (for query objects and base objects that require the most query execution time). For some algorithms, an exponential dependence manifests itself in terms of an "intrinsic dimension" (see references in [3]) that can be smaller than the "external" dimension D.

It is possible to overcome (or more exactly, to bypass) the curse of dimensionality by transition from exact to a faster but approximate search admitting a distinction of its results from those of exact search. The reduction of some other problems to the approximate near neighbor problem is described in [12].

1.2.3. Approximate similarity search queries. Let us consider widespread types of approximate similarity search queries [14–16, 12].

Denote by (c, δ)-rNN1 a randomized c-approximate r-near neighbor query; with probability no smaller than $1-\delta$, it returns any base object whose distance from the query object does not exceed cr ($c>1$) if there is a base object whose distance from the query object does not exceed r. If such an object is absent, then an object is returned whose distance does not exceed cr or the answer "no."

Denote by (c)-rNN1 a (c, δ)-rNN1 query with $\delta = 0$ (the deterministic version).

Denote by (δ)-rNN an all randomized r-near neighbors query; it returns all base objects whose distance from the query object does not exceed r with probability no smaller than $1-\delta$ for each such an object.

Denote by c-NN a c-approximate nearest neighbor query; it returns a base object whose distance from the query object is no more than by a factor of $c>1$ longer than the distance to the exact nearest neighbor.

The execution of a $c(1+\gamma)$-NN query (for some $\gamma > 0$) can be reduced [16, 12] to a sequence of (c)-rNN1 queries with different radiuses. Let Δ be the finite (bounded) ratio of the largest to the smallest distances between the base objects (including the query), and let the smallest distance be equal to 1. We construct $O(\log_{1+\gamma} \Delta)$ IS instances for (c)-rNN1 with radiuses r assuming values $0.5(1+\gamma)^i$ for $i = 0, 1, \ldots$. After arriving a query and performing binary search in these ISs (i.e., after executing $O(\log \log_{1+\gamma} \Delta)$ (c)-rNN1 queries with different r), we find an answer to a $c(1+\gamma)$-NN query as the object returned for the smallest value of r. For arbitrary metric spaces (with unlimited Δ), a more efficient (but impractical) reduction based on an IS is proposed in [16] for a (c, δ)-rNN1 query. To avoid the increase in memory cost in searching for an approximate NN, other ISs without specified theoretical guarantees are frequently used (their quality is checked on bases).

Note that the notation NN in (c)-rNN1, (c, δ)-rNN1, and (δ)-rNN queries is used for a near neighbor and that in NN, kNN, c-NN, and $c(1+\gamma)$-NN queries is used for the nearest neighbor. Practical ISs executing some randomized queries, which are described in this subsection, with guarantees for execution time for worst-case data are considered in Sec. 2.

1.2.4. Quality measures of similarity search. Approximate similarity search is required in practice since, for many applications, it suffices to obtain even approximate results but quickly.

For algorithms of approximate search with quantitative guarantees, two types of distinctions of their results from the results of exact search are considered. In the case of deterministic approximation algorithms, the distance to the found objects no more than by a given multiplier exceeds the distance to the objects that are the exact answer to the query. In the case of randomized Monte Carlo algorithms, false negatives are allowed, namely, an algorithm does not necessarily return objects that are answers to the query (but the probability of this over all implementations of the algorithm and its search time in analysis are not bounded for worst-case data). Monte Carlo algorithms executing randomized approximate queries can give both distinctions. Randomized Las Vegas algorithms are performed without false negatives and, in analysis, restrict the search time averaged over all implementations of an algorithm for worst-case case (rather than for the average as was stated in [4]).

In practice, users are frequently interested not in theoretical guarantees but in the execution time of queries in experiments on concrete bases. This time usually depends on the choice of IS parameters. For exact search, the best is IS providing the smallest time cost. For approximate search, the choice of IS parameters usually makes it possible to achieve a trade-off between the quality of search results (the degree of their distinction from the results of exact search) and search time. Therefore, ISs for approximate search are compared by (averaged) query execution time for a fixed search quality or by the value of a search quality measure for the same search time. An important characteristic also is IS memory cost (quadratic cost is unacceptable).

The following well-known measures [17–19] of the quality of executing a concrete query are often used: recall equal to $n1/n2$ and precision equal to $n1/n3$, where $n1$ is the number of returned objects that coincide with base objects

relevant to the query, $n2$ is the number of base objects relevant to the query, and $n3$ is the number of returned base objects. For a kNN query, $n2 = n3 = k$, and, therefore, precision is equal to recall. As relevant objects, the objects are used that are returned by exact search queries or are specified by an expert. "Good" neighbors can be not only exact neighbors but also base objects close to exact neighbors. Therefore, search quality is also measured by the ratio of the sum (or the average) of the distances of the query object to the returned objects to the corresponding value for the reference result (distance ratio metric), etc. [19].

When executing many queries, the quality measure values obtained for individual queries are averaged. In particular, for an NN query, recall (equal to precision) is measured as the percentage of the queries for which the reference NN [20] is returned.

1.3. General principles of construction and applications of index structures. In both parts of this survey (Part II is being prepared for publication), ISs are mainly considered in construction of which the set of base objects is somehow divided into subsets (in some types of ISs, they do not intersect and, in others, their intersection is possible). Some types of ISs store and use information on domains to which belong these subsets in the space of objects. Sometimes, several ISs with different partitions into subsets are constructed for one base.

There are static and dynamic ISs. In static ones, the algorithm for constructing an IS presumes the presence of the whole base of objects. Dynamic ISs are constructed by operations of inserting objects in the process of arriving of data.

When executing a query, procedures are used that may be considered as variants of the two-stage filter-and-refine (F&R) strategy. At the first stage, candidate objects are quickly selected. The results of the first stage are refined at the second stage (usually using linear search among candidate objects based on the distance/similarity specified in the query). The F&R strategy is sometimes applied repeatedly in executing one query. If the total number of candidate objects is less than the number of base objects N and they are rather quickly selected, then an acceleration of executing a query with respect to linear search can be obtained (sublinear search).

For exact search in executing a query, candidates are chosen so that they guaranteedly contain an answer to an exact search query. Many ISs of exact similarity search use varieties of the branch and bound (B&B) method (see [3]). However, sublinear execution time of exact search queries is not guaranteed and not observed in practice (Sec. 1.2.2).

In the basic version of IS for approximate similarity search based on hashing preserving similarity, the hash values are associated with objects such that the probability of coincidence of hashes of similar objects is higher than that of hashes of dissimilar objects. Objects with identical hash values form subsets in IS. In executing a query, candidate objects are subsets with the same hash value as the query object. This provides the retrieval of candidate objects similar to the query object and guarantees sublinear execution time for some types of randomized approximate search queries (in particular, (c, δ)-rNN1).

2. INDEX STRUCTURES BASED ON HASHING PRESERVING SIMILARITY

In locality-sensitive hashing (LSH), unlike the usual one, LSH functions generate hash-values so that the probability of their coincidence for similar objects is higher than for dissimilar ones ([15, 16, 2, 4]). Objects with the same hash form a bucket. When a query object arrives, its LSH hash is generated, base objects are extracted from the corresponding bucket, and they are used as candidates for linear search based on the initial distance. The hash-values of similar objects generated by one LSH function can turn out to be different and, hence, to increase the probability of finding objects close to the query object, lists of candidates from buckets of several independent LSH functions are united by analogy with forests of trees (see, for example, [20]). LSH-based index structures are among the few practically implemented ones for some types of approximate randomized similarity search queries with guarantees of sublinear time in the worst case.

2.1. Index LSH structure for approximate similarity search.

2.1.1. LSH functions. The family F of hash functions h is (r_1, r_2, p_1, p_2)-sensitive for any objects a and b under the following conditions: if $\operatorname{dist}(a, b) \le r_1$, then $\Pr_F\{h(a) = h(b)\} \ge p_1$, and if $\operatorname{dist}(a, b) \ge r_2$, then $\Pr_F\{h(a) = h(b)\} \le p_2$ [15, 16], where the probability Pr corresponds to a random uniform choice of hash functions h from the family. For a useful LSH family, $r_1 < r_2$ and $p_1 > p_2$. This definition of LSH is based on the gap of distances between similar and dissimilar objects. By analogy, LSH can be defined based on the gap of similarities [21].

In another definition of LSH based on the monotonicity of the similarity value [22], $\Pr_F\{h(a) = h(b)\} = \operatorname{sim}(a, b)$ must hold, where sim is the similarity measure of objects that assumes values in $[0, 1]$. Note that sim can also be a monotonically increasing function (with its values in $[0, 1]$) of some other similarity function [21]. There also is a similar definition for distances. The definition of LSH based on monotonicity implies a gap-based definition [21]. We mention

the generalization of LSH of the form $\Pr_F\{h_1(a) = h_2(b)\} = f(\mathrm{dist}(a, b))$, where $f(\,)$ is not necessarily a decreasing function of dist [23].

Example of an LSH function. For an angle θ between vectors, the following family SimHash [22] of LSH functions is widely used: $h_{\mathbf{r}}(\mathbf{a}) = \mathrm{sign}(\langle \mathbf{r}, \mathbf{a}\rangle)$, $\mathbf{r}$ is a random vector of length D with components that are independent and identically distributed (i.i.d.) random quantities (r.q.) from the Gaussian distribution ($\mathbf{r} \sim \mathrm{Norm}(\mathbf{0}, \mathbf{I}_D)$), $\mathrm{sign}(z) = 1$ when $z \geq 0$, and $\mathrm{sign}(z) = -1$ when $z < 0$. For this LSH function, $\Pr\{h(\mathbf{a}) = h(\mathbf{b})\} = 1 - \theta(\mathbf{a}, \mathbf{b})/\pi$. Other examples of LSH are considered in Sec. 2.2.

2.1.2. Basic index LSH structure. When constructing an LSH-based IS, L LSH functions g are created and stored, each of which is the concatenation of m LSH functions h randomly chosen from the corresponding LSH family. A function g is (r_1, r_2, p_1^m, p_2^m)-sensitive. (In analyzing an (c, δ)-rNN1 query, it is assumed that $r_1 = r$ and $r_2 = cr$.) For each of L functions g, its own hash table is constructed. Each base object y is placed in one bucket of each of L hash tables (the bucket that corresponds to its hash $g(y)$ for this table). Thus, the objects with the same values of g are placed in one bucket. Since the majority of cells of tables are empty when $N << 2^m$, usual hashing is additionally applied to LSH hashes. Other features of implementing the basic LSH IS are described in [24].

To execute a query, the F&R strategy is used in LSH IS. Hashes of query object are generated by the same LSH functions that are used for constructing IS. Candidate objects are extracted from the buckets whose hashes coincide with the hashes of the query object (a collision). To execute a (c, δ)-rNN1 query, a fixed number of objects are taken from these buckets, for example, $3L$, including duplicates ("Strategy 1" [15]). For a (δ)-rNN query, all objects from these buckets are used ("Strategy 2" [15]). The final result is obtained by linear search in the list of candidates by the distance/similarity measure of the query.

Using $\rho = \log 1/p_1 / \log 1/p_2 = \log p_1 / \log p_2 \in (0, 1)$, we can write $L = N^{\rho}$. We obtain the following memory cost for an LSH structure consisting of L tables and the base: $O(DN + N^{1+\rho})$. The search time consists of the time of computation of hash functions $O(Lm) = O(N^{\rho} \log N)$ and the time of computation of distances to candidates $O(LD) = O(N^{\rho} D)$. Strategy 1 guarantees the mentioned search time since the number of candidates is bounded. Thus, $\rho < 1$ (sensitivity) characterizes the "quality" of LSH functions (the acceleration of search with respect to linear search and memory cost). Usually, $\rho = 1$ for $c = 1$ and $\rho \to 0$ as $c \to \infty$.

When calculating the parameters for Strategy 2 (a (δ)-rNN query), $L = \lceil \log \delta / \log(1 - p_1^m) \rceil$ is chosen [24], and the value of m that minimizes query time is determined in experiments. B [25], L is chosen proceeding from existing memory, and $m = \lceil \log(1 - \delta^{1/L}) / \log p_1 \rceil$.

2.2. Examples of LSH functions. The calculation of LSH IS for a concrete distance/similarity measure requires the presence of an LSH family for it (with known characteristics that do not depend on a concrete collection of base objects). An arsenal of LSH families is considered in [15, 26, 2, 27]. Most LSH families are developed for vectors (but see [28, 29] for kernel similarities).

For spaces L_s, $s \in (0, 2]$, LSH families [24] are proposed that use vectors $\mathbf{r}$ with components, i.i.d. r.q., from s-stable distributions $h(\mathbf{a}) = \lfloor (\langle \mathbf{r}, \mathbf{a}\rangle + U)/w \rfloor$, where w is the parameter determining the size a hash bucket and U is an i.i.d. r.q. from a uniform distribution in $[0, w]$: $U \sim \mathrm{Unif}(0, w)$. Examples of s-stable distributions are the Cauchy distribution for $s = 1$ and Gaussian distribution for $s = 2$. For these families, $\rho = \max\{1/c^s, 1/c\} + o(1)$. Denote LSH functions of this family by L2LSH for L_2 and L1LSH for L_1.

In [30], for L_2, $\rho = 1/c^2 + o(1)$ is obtained, which is an optimal value. Initial vectors are first projected randomly (see [1]) into vectors of reduced dimension, and then a set of random vectors is chosen. The value of an LSH hash is the identifier of the first random vector whose spherical neighborhood with a given radius contains the vector being hashed. In more practical versions, the hash value is the nearest lattice point [30, 31].

In order for a distance have a family of LSH functions, it is necessary (but not sufficient) that it be metric [22] and have an isometric embedding into L_1 [32]. If each of two similarities has an LSH function, then their product also has it. A necessary and sufficient condition of preserving the LSH property for a function of transformation of similarity is its membership in the class of (scaled) probability generating functions. In [33], similarities that do not have LSH are

considered, and the concept of a distortion is introduced to characterize the approximation of these similarities by similarities that have LSH. Dense upper and lower bounds of distortion are shown and confirmed by experiments for well-known similarities without LSH.

Note that the presence of LSH families for L_1 makes it possible in principle to use LSH IS for approximate search based on representations of objects and their distances that can be transformed into L_1 with some distortion (for example, edit distance, the earth mover's distance, etc. [34, 16, 35, 1]). LSH functions can also be used for the formation of binary or integer-valued components of vectors (sketches) to estimate distances/similarities (corresponding to LSH functions) between initial objects from the empirical probability of the coincidence of hashes ($\mathrm{dist}_{\mathrm{Ham}} / D$ of sketches [2]) according to the definition of LSH based on monotonicity.

As an example LSH, we will consider recently appeared LSH functions for a unit sphere.

2.2.1. LSH functions and implementation of search based on the Euclidean distance of unit vectors. An optimal and efficiently implementable family of LSH functions for the Euclidean distance of vectors on a unit sphere is analyzed in [36]. Its applicability to searching based on some other measures follows from the relations presented in Sec. 1.1. The family uses randomly rotated hyperoctahedrons (unit spheres in L_1) [37, 38]. To obtain the value of the LSH hash of a vector, it is multiplied by a random i.i.d. Gaussian matrix $R\,(D \times D)$, which is specific for each LSH function, and the result is normalized to unit norm. Then the hyperoctahedron vertex nearest to the obtained vector (maximum vector component) is found, and it is its number that is the hash value. This is computationally efficient since the distance is equivalent to the scalar product, and there is only one nonzero component in vectors of vertices.

For the complete range of distances $0 < r < cr < 2$, the dependence $\rho = (1/c^2)(4 - c^2 r^2)/(4 - r^2) + o(1)$ is obtained, where $o(1) \to 0$ as $D \to \infty$. When $r = \sqrt{2}/c$ and $rc = \sqrt{2}$, this LSH reaches its optimal value $\rho = 1/(2c^2 - 1)$ as well as LSH dependent on data for L_2 [39] (which, however, are practically not realizable because of a large computation time, Sec. 2.6.3). The lower bound ρ is also given for different relations p_1 and p_2, which shows that the hyperoctahedral LSH is close to optimal.

To accelerate hashing (without theoretical guarantees) [36], the hashing tricks of dimensionality reduction and high-speed multiplication by an orthogonal matrix pipeline (see [1]) and also a specialized multi-probe LSH (Sec. 2.3.1) are used. For a number of bases, a software implementation of FALCONN accelerates an NN query in comparison with SimHash by a factor of 4–10 for large N [36] (though the obtaining of the value of a hash requires more time). However, note that SimHash can also be accelerated by using fast matrix pipelines [2].

In [40], a hypercube LSH is investigated in which a partition into domains is created by orthogonal hyperplanes (Voronoi domains around hypercube vertices). A random rotation of the vector being hashed is preliminary executed (for fast pseudorandom rotation, theoretical guarantees can also be obtained [41]). For hypercube LSH, ρ is located between SimHash and hyperoctahedral one, but some applications faster operate with hypercube LSH [40].

2.3. Improvements of basic LSH IS.

2.3.1. Multi-probe LSH. A great number L of LSH tables requires considerable memory cost. To decrease L (with preservation of search quality), it is proposed in [42, 43] to change the procedure of executing a query so that to extract candidates not only from one bucket of the table corresponding to the LSH hash of the query but also from other buckets "close" to the query object. This "multi-probe" LSH uses both modifications of the query vector [42] and (faster) modifications of the LSH hash of the query [43]. The scheme of generation of modifications and sequence order of traversal of buckets is constructed in such a way that it is computationally efficient and, first of all, provides visits to the most "promising" buckets for a concrete query object. In [43], the same precision and time of a kNN query are obtained as for the basic LSH with decreasing L by a factor of larger than 10.

Experimental investigations of versions of modifying the hash code of a query [44, 45] showed good results. In experiments [36] on searching for an (exact) NN with the same $1 - \delta$ and L, the query time for the multi-probe variant has decreased more than 10 times (at the cost of a larger optimal value of m providing a smaller number of candidates).

Multi-probe LSH is also used in theoretical works. In [46], it is applied to obtaining a smooth trade-off between memory cost and search time. For the multi-probe variant of adaptive LSH IS [47] (and Sec. 2.3.3), for a (δ)-rNN query, not only an economy of memory but also an acceleration of executing a query are theoretically proved.

For an rNN query (without false negatives) in L_s, $s \in [1, \infty]$, a family of LSH functions of the form $h_s(\mathbf{x}) = \lfloor \langle \mathbf{x}, \mathbf{r} \rangle / (rD^{1-1/s}) \rfloor$ is proposed [48], where r is the radius of a query and $\mathbf{r}$ is a vector of an i.i.d. r.q. from

a bounded centered distribution, $-1 \le r_i \le 1$, $\mathrm{E}\{r_i\} = 0$. Here, as candidates for a query $\mathbf{x}$, it is required to return all base objects $\mathbf{y}$ for which $||g(\mathbf{x}) - g(\mathbf{y})||_\infty \le 1$. To this end, it is proposed to use 3^m modifications of the hash of a query or 3^m modifications of the hash of each base object. Unfortunately, candidates can be objects with a long distance to the query (up to cr, $c \ge \max\{D^{1/2}, D^{1-1/s}\}$). The acceleration of a query is considered in [49].

2.3.2. Models and choosing parameters of LSH IS for a concrete base. Taking into account features of data of a concrete base, it is possible to improve characteristics of search in LSH IS (see also Sec. 2.6.3). In [24], parameters are chosen by an exhaustive search of their combinations on a software implementation of a real LSH structure. This is a computationally intensive problem even if a subset of a base is processed. To solve this problem [50, 45], statistical models of LSH IS are proposed for L2LSH and randomized (k)NN queries. The models use distributions of distances between a query and (k)NN and also between base objects.

In the multi-probe LSH model from [50], to maximize the recall of search, the maximum L is chosen proceeding from available memory. With a given average recall value of search on the base, w in $h(\mathbf{a}) = \lfloor (\langle \mathbf{r}, \mathbf{a} \rangle + U)/w \rfloor$ and the number of m concatenated functions are offline adjusted to minimizing the query time (the average percentage of candidates among base objects). For a concrete query object based on distances to the current kNNs, the number of modifications of the query is adjusted to achieve the prescribed recall of search. A detailed model from [45] minimizes the average time of searching for the prescribed probability of an error smaller than δ (with a simplified query modification without adaptation to the query) and can be used for adjusting parameters to different types of queries.

2.3.3. Adaptation to a concrete query. Adaptation to a concrete vector query was considered in the model of multi-probe LSH from [50]. In [31], it is proposed to improve the trade-off between the search speed and accuracy (in L_2) by choosing the LSH tables out of their existing collection in which the vector query with the highest probability will be placed into one bucket with the nearest neighbor when executing the query. The relevance criterion of a hash bucket, which is based on the query distance to the center of the bucket, is used.

In [47], a (δ)-rNN query is considered. For the basic LSH IS and "difficult" queries, the average number of candidates can amount to $O(nL)$, where n is the number of base objects that are answers to the query. To solve this problem in multilevel adaptive LSH [47], collections (levels) of LSH tables are constructed for different combinations of m and L (L increases with increasing m). When executing a query in this IS, the average number of candidates $\Theta(n(N/n)^\rho)$ is always less than N, and the IS variant with a query modification brings it closer to the lower bound $O(N^\rho + n)$. Large memory cost is required to implement this approach.

In the practical algorithm from [25], for a (δ)-rNN query, LSH IS is constructed for a fixed L by choosing $m = \lceil \log(1-\delta^{1/L}) / \log p_1 \rceil$ (see Sec. 2.1.2, Strategy 2). In order to choose the use of the basic LSH IS or linear search in executing a concrete query, it is proposed to quickly estimate the query time for this LSH. To this end, in HybridLSH [25], a Hyperloglog structure [51] is associated with each hash bucket, which makes it possible to approximate the number of objects without duplicates in L buckets chosen by the query hash.

2.3.4. Excluding candidates based on LSH hashes. The probability estimates for the coincidence of LSH hashes of objects can be used for fast processing of candidates to exclude the candidates for which the probability that the similarity excesses a threshold value is small [52]. A decision on the exclusion can be made after estimating a small part of hashes out of their total number. For example, if there are 1000 hashes and only 10 out of the first 100 ones coincide, then it is very improbable that the similarity of objects exceeds 0.8. Moreover, the magnitude itself of similarity can be estimated with the precision preassigned by the user. The probability distribution functions for the similarity value that are required under the condition of the coincidence of the preassigned percentage of hash-values are obtained in [52] for LSH MinHash [27] (Jaccard similarity) and SimHash [22] (with a modification for cosine similarity). An application to (normalized) kernel similarity functions is considered in [52].

2.4. Searching for approximate nearest neighbors in LSH index structures. A theoretically substantiated basic LSH IS for a (c, δ)-rNN1 query requires the construction of a collection of tables with their parameters for each collection of r and c. When executing a randomized c-NN query by a sequence of (c, δ)-rNN1 queries with different radiuses (see Sec. 1.2.3), this leads to large memory cost. To save memory, algorithms are proposed that emulate queries with different radiuses in one IS based on LSH.

In an LSH forest [53], the number m of the functions h being concatenated is a variable such that base objects should assume different values of LSH hashes. Instead of hash tables, an experimentally chosen number L of prefix trees is used in which to objects correspond leaf nodes. The increase in the radius of a query is emulated by the choice of candidate objects

whose hashes partially coincide with the query hash. The leaf with the largest prefix coinciding with the query in each tree is first found. Then a synchronous level-by-level traversal of L trees is performed from the lowermost level to the root with extracting leaves-descendants of parent nodes until the number of candidates reaches the preassigned one. Among the candidates, k candidates nearest to the query are chosen. The theoretical analysis in [54] considers an LSH forest variety as practical version of LSH (Sec. 2.6.3) dependent on data for a (c, δ)-rNN1 query (with a smaller value of ρ than for LSH independent of data).

In the LSB tree [55] for searching in L_2, dimension is first reduced by randomly projecting using a Gaussian i.i.d. matrix. Vectors are transformed into values of Z-order [56], and base vectors index a B-tree [57]. A single tree or a forest of trees is used. The execution of a series of queries with an increasing radius is emulated by the retrieval of vectors from leaf nodes in decreasing order of the number of coinciding more significant bits of Z-values.

SKLSH [58] uses keys that are concatenation of values of L2LSH hashes. The metric distance between keys is proposed (based on the coincidence of prefixes and distinction in the next component) whose value reflects the distance L_2 between vectors. This allows one to order keys in constructing IS so that keys of base vectors close to the query vector are stored nearby in memory. Using B+tree [57] allows one to first extract (during searching) all the objects whose keys are closer to the key of the query object. An advantage over LSB tree and C2LSH (see below) is reached owing to extracting a larger number of candidate objects using a much smaller number of random input-output operations. In [59], early stopping of SKLSH is used for acceleration.

In [60], it is proposed to rank candidates by the frequency of their occurrence in the list of candidates. In FBLSH [61] and C2LSH [62], as candidates, the base objects are chosen for which the number of coincidences of individual LSH hashes h with LSH hashes of the query exceeds a threshold value. A similar selection is used in [63, 64].

In C2LSH [62], the increase in search radiuses $\{1, c, c^2, \dots\}$ is used with the help of modifications of hash functions in the capacity of which an L2LSH version is used. The modification is carried out as $\lfloor h(\cdot)/c \rfloor$, where $h(\cdot)$ is the value of the hash function for the previous radius.

In LazyLSH [63], queries are executed for distances L_s, $s \in [1/2, 1]$, with using a C2LSH version for L1LSH. The radius r for L_s, $s \in [1/2, 1]$, is approximated through the radius r for L_1. In contrast to C2LSH where the increase in r leads to the increase in the size of buckets that is independent of the query object, buckets adjacent to the bucket of the query object are combined in LazyLSH, which increases the probability of the coincidence of hashes for base vectors close to the query. When simultaneously executing a query for L_s with different s, it is possible to optimize the total time of extracting candidates. This approach can also be used with other ISs for L_1.

In QALSH [64], in constructing IS for L_2, a partition into buckets is not used, but the values of projection of $\langle \mathbf{y}, \mathbf{r} \rangle$ base vectors $\mathbf{y}$ onto random directions of $\mathbf{r}$ are indexed with the help of B+tree. With arriving a query $\mathbf{x}$, base objects in a virtual bucket are searched for in B+tree in the range $[\langle \mathbf{x}, \mathbf{r} \rangle - wr/2, \langle \mathbf{x}, \mathbf{r} \rangle + wr/2]$. Owing to the absence of a fixed partition into buckets, queries with radiuses r from $\{1, c, c^2, \dots\}$ can be executed with any $c > 1$. In experiments, QALSH exceeds C2LSH and LSB forest.

In [65], candidate objects are extracted in increasing order of $\mathrm{dist}_{\mathrm{Ham}}$ of their binary LSH hashes from the hashes of the query object (up to the achievement of the number of candidates that is empirically determined for a problem and for k). To search by $\mathrm{dist}_{\mathrm{Ham}}$, the MIH structure [66] is used (executing queries with increasing radius). Binary values of L2LSH hashes are obtained by choosing the parameter w in $h(\mathbf{a}) = \lfloor (\langle \mathbf{r}, \mathbf{a} \rangle + U)/w \rfloor$. This BLSH returns more precise kNNs for a less time than LSB, C2LSH, and SKLSH, especially for data of higher dimension ($D = 192$).

In contrast with the considered approaches, it is proposed in Selective Hashing [67] to construct a collection of LSH ISs for each of search radius from $\{r, cr, c^2 r, \dots\}$, however, to place an object only into the table bucket for one of radiuses. Base objects from space domains of high density are placed into smaller buckets (corresponding to smaller radiuses of the query). This can be considered as fine tuning of IS locally to each base object. When executing a query, candidates are extracted from tables starting with a small radius until the remained ISs still can include potential kNNs. Since the stop condition takes into account the position of the query object with respect to base objects, adaptation to the query takes place. This practical approach provides memory cost at the level of the basic LSH IS for one radius and the recall of kNN search at the level of LSH IS for a set of (increasing) radiuses.

2.5. Locality-sensitive filtering and time/memory trade-off. The basic LSH IS operates in the following symmetric mode: the value of ρ (see Sec. 2.1.2) is the same for both query time and memory cost (and the time of insertion/deletion of objects). However, some applications require the asymmetric mode with different trade-offs between the increase in memory cost and decrease in query execution time. A smooth time/memory trade-off providing sublinear search time in L_2 in the worst case for any $c > 1$ and improving results [46] for all c is obtained in [68] based on the locality-sensitive filtering (LSF) approach [69].

LSH ISs associate a bucket with each LSH function value and thereby provide the following exhaustive partition of the space of objects: each object is placed into one of buckets. An LSF filter differs in that only a part of input objects passes through it. Similar objects have a higher probability to pass through an LSF filter than dissimilar ones. LSF filters can be used individually or can be combined sequentially and in parallel. In [70], to fast emulate a large number of LSF filters, it is proposed to obtain objects at the output of some set of basic filters, and then to find the result at the output of one emulated filter by the intersection of the objects of the subset of basic objects that corresponds to it.

A bucket stores the objects remained after passing an LSF filter or their collections on the path to the bucket. The same object can belong to different buckets. In executing a query, objects from the buckets into which the query object has passed become candidates. Parameters of filters in constructing an IS and in executing a query can differ, which provides the mentioned trade-off.

When creating LSF filters for searching on the unit sphere S^{D-1}, the idea of partitioning the sphere into domains by a large number of small spherical segments (spherical caps (SCs)) is used. A vector **a** belongs to an SC (r,t) if $\langle \mathbf{r}, \mathbf{a} \rangle \geq t$, where **r** is a random Gaussian vector or a random vector with S^{D-1}. (Note that a similar transformation is used for obtaining sparse binary/ternary vectors in [71–74].) A threshold t_u used in constructing IS can differ from t_q used in processing a query.

If the number of SCs is large, then the determination of the SCs to which a vector belongs is computationally intensive. To solve this problem, to each domain-bucket corresponds the intersection of several SCs in [68]. To determine the bucket to which an object belongs, a tree with $A+1$ levels is used in which each node corresponds to an SC. The coefficient of branching does not exceed M and, therefore, at the Ath level, there are no more than M^A leaves. Each node stores one random i.i.d. vector from Norm $(\mathbf{0}, \mathbf{I})$. When constructing IS, the set of vectors-objects of some node is divided by its sons into M (possibly, intersecting) subsets each of which is determined from $\langle \mathbf{r}, \mathbf{y} \rangle \geq t_u$. For nodes with a nonempty set of objects, the partition is repeated until the Ath level of the tree is reached.

The processing of a vector query begins with the root and proceeds to the sons for which $\langle \mathbf{r}, \mathbf{x} \rangle \geq t_q$. Candidates are vectors from nodes of the Ath level reached by the vector query. The choice of the parameters A, M, t_u, and t_q depends on r and c and also on ρ_u and ρ_q. If $t_u = t_q$ then $\rho_u = \rho_q$ and the LSH mode is obtained. This IS is not implemented in practice.

In [69], an implemented algorithm is proposed. To a leaf corresponds a spherical segment, but the vector determining it consists of $A = \Theta(\log D)$ vector parts of dimension D/A. For each part, M vectors with S^{D-1} are randomly generated and are normalized to provide the unit norm of the full vector. Out of vector parts, M^A different vectors **r** of dimension D that correspond to buckets can be composed. It is shown that the probability of placing vectors into a bucket with a compound **r** is close to the probability for a random **r** with S^{D-1}.

To fast determine the bucket containing the input vector, IS is constructed in the form of an A-level tree with branching coefficient M in which a random vector part corresponds to each node. For each of A parts of the input vector, M scalar products with vectors of nodes are computed, which makes it possible to quickly determine buckets (corresponding to leaves) that contain the input vector (the complexity of computations is A times the number of buckets that contain the vector). The values of t_u and t_q can differ.

In the symmetric mode $\rho_u = \rho_q$, such an IS allows one to obtain smaller values of ρ than those for spherical and hyperoctahedral LSHs (see Sec. 2.2.1) for data with $D = O(\log N)$. For the worst case, these ISs independent of data allow one to reach a flexible time/memory trade-off $(c^2+1)\sqrt{\rho_q} + (c^2-1)\sqrt{\rho_u} \geq 2c$. The extension from S^{D-1} to the entire L_2 (Sec. 2.6.1) allows one, for all $c>1$, $r>0$, and ρ_u and ρ_q related to this expression, to obtain IS for a (c,δ)-rNN1 query in L_2 with memory cost $N^{1+\rho_u+o(1)}$ and query time $N^{\rho_q+o(1)} + DN^{o(1)}$. For $\rho_u = \rho_q$, we obtain $\rho \geq 1/c^2$. The extension to other L_s is considered in Sec. 2.6.1. ISs dependent on data allow one to improve characteristics (Sec. 2.6.3).

2.6. Some theoretical aspects and adaptation to data.

2.6.1. Extensions to L_s. The results of analysis of ISs for searching in S^{D-1} are extended to the entire Euclidean space using the reduction of L_2 to S^{D-1} [75] that preserves the ratio of L_2 distances (with a multiplicative distortion).

To extend the results obtained for L_2 to L_s, a deterministic component-wise embedding [76] (with multiplicative and additive distortions) from L_s $(1 \leq s < 2)$ into $L_2^{2/s}$ is used (with the corresponding change in r). The extensions of the basic LSH IS in L_2 for which $\rho = 1/c^2 + o(1)$ (see [30] and Sec. 2.2) to L_s give the value of $\rho = 1/c^s + o(1)$ for L_s [12].

To extend the results of analysis of ISs for approximate search by the scalar product of vectors from S^{D-1} (which is equivalent to the search by the Euclidean distance, see Sec. 1.1) to L_s ($0< s \le 2$), it is proposed [70] to use an approximate embedding of the kernel similarity of D-dimensional vectors $\kappa(\mathbf{a}-\mathbf{b}) = \exp(-||\mathbf{a}-\mathbf{b}||_s^s)$, $0< s \le 2$, into the scalar product of vectors on S^{D-1} ([77] and [1, 2]). Then the execution of a (c,δ)-rNN1 query in L_s is equivalent to the execution of a similar query by $\mathrm{sim}_{\mathrm{dot}}$ for vectors with S^{D-1} with similarity value $\exp(-r^s)$ (adjusted for a distortion of the embedding).

Note that the reductions of this section are absent in practical ISs.

2.6.2. Boundaries. An analysis of the proposed similarity search algorithms (that are not always realizable in practice) allows one to obtain upper bounds for memory cost and query time. Lower bounds for real-valued vectors with distance L_s are usually obtained through proved (for some "difficult" data distributions) lower bounds for searching for binary vectors $\{0,1\}^D$ by $\mathrm{dist}_{\mathrm{Ham}}$ using $\mathrm{dist}_{\mathrm{Ham}}(\mathbf{a},\mathbf{b}) \equiv ||\mathbf{a}-\mathbf{b}||_s^s$. The use of L_s^s instead of $\mathrm{dist}_{\mathrm{Ham}}$ transforms c into c^s in the expression for ρ [12]. For a more general case of real-valued vectors, the lower bound cannot be smaller than for binary ones.

For the basic LSH, the lower bound for the Hamming distance $\rho \ge 1/c - o_D(1)$ [78] implies $\rho \ge 1/c^s - o_D(1)$ for L_s. For a wide class of data-independent ISs "list-of-points" (which includes both LSH and LSF), for binary vectors with $D = \omega(\log N)$, the lower bound is shown [68] for the entire range of ρ_u, ρ_q: $c\sqrt{\rho_q} + (c-1)\sqrt{\rho_u} \ge \sqrt{2c-1}$ with replacing c by c^s for $1 \le s \le 2$. Accurate within addends $o(1)$, this lower bound ρ corresponds to the upper bound for the partitions dependent on data (Sec. 2.6.3). For $\rho_u = \rho_q$ and L_1, we obtain $\rho = 1/(2c-1)$, and, for L_2, we have $\rho = 1/(2c^2-1)$, which also corresponds to lower bounds for LSH dependent on data. An analysis of the lower bound for $s > 2$ is absent at present.

2.6.3. Adaptation to data. In practice, index structures that adapt to data of a base often exceed ISs designed independently of data though mainly do not give worst-case guarantees. This is also true for similarity search by representations created by learning. Reviews of ISs and formation of representations with learning are presented in [26, 79–81].

In [82], a theoretical analysis of decreasing query time in comparison with the worst case and the choice of parameters for LSH that is independent of data and manipulates with well-dispersed data or with data of a low intrinsic dimension is presented.

The adaptation of IS to data of a base, which makes it possible to improve characteristics of IS for worst-case data is considered in [83, 39, 68, 54]. It turns out that, for some ("pseudorandom") configurations (arrangements) of base objects, ISs independent of data can be constructed with some ρ smaller than for worst-case data. The objects of the worst-case base are divided into subsets that are reduced to such configurations.

In [83], an improvement of ρ for LSH for L_2 is obtained for sufficiently large c, and, in [39], the results are improved up to the theoretically optimal [84] $\rho = 1/(2c^2-1) + o(1)$ for all c. In [68], the following trade-off is obtained (optimal for a wide class of ISs) between ρ_u and ρ_q: $c^2\sqrt{\rho_q} + (c^2-1)\sqrt{\rho_u} \ge \sqrt{2c^2-1}$. Unfortunately, as a result of the complexity of reducing worst-case data to "pseudorandom" configurations, the methods from [83, 39, 68] are not implemented in practice. In [54], an analysis of a practical scheme (a modification of an LSH-forest) is presented whose characteristics for worst-case data exceed those of LSH independent of data (though they are inferior to optimal bounds [39]).

CONCLUSIONS

An advantage of the ISs and their modifications considered in this article and based on locality-sensitive hashing consists of a theoretically substantiated choice of IS parameters, which allows one to guarantee the execution time sublinear with respect to the number of base objects for some types of queries of worst-case approximate similarity search. The second part of this survey paper will discuss and compare this approach with ISs based on the explicit use of domains (treelike, etc.), neighborhood graphs [3], transformations of initial (real-valued vector) representations of objects, and autoassociative memory [85]. Note that the well-parallelized algorithm FLASH [86] specialized for the Jaccard similarity between binary sparse vectors of very high dimension (a modified version of LSH IS using MinHash with one permutation for obtaining a large number of hash-values [87], a bucket of bounded size due to samplings, common buckets for different LSH tables, and ranking of candidate objects by the frequency of their occurrence without computing initial distances/similarities to the query object) has showed an advantage over all other ISs.

The author is grateful to Alex Andoni for explanations of some aspects of his investigations.

REFERENCES

1. D. A. Rachkovskij, "Real-valued embeddings and sketches for fast distance and similarity estimation," Cybernetics and Systems Analysis, Vol. 52, No. 6, 967–988 (2016).
2. D. A. Rachkovskij, "Binary vectors for fast distance and similarity estimation," Cybernetics and Systems Analysis, Vol. 53, No. 1, 138–156 (2017).
3. D. A. Rachkovskij, "Distance-based index structures for fast similarity search," Cybernetics and Systems Analysis, Vol. 53, No. 4, 636–658 (2017).
4. D. A. Rachkovskij, "Index structures for fast similarity search for binary vectors," Cybernetics and Systems Analysis, Vol. 53, No. 5, 799–820 (2017).
5. C. Manning, P. Raghavan, and H. Schütze, Introduction to Information Retrieval, Cambridge University Press, New York (2008).
6. R. Datta, D. Joshi, J. Li, and J. Wang, "Image retrieval: Ideas, influences, and trends of the new age," ACM Computing Surveys, Vol. 40, No. 2, 1–60 (2008).
7. M. M. Fouad, "Content-based search for image retrieval," Int. J. Image, Graphics and Signal Processing, Vol. 5, No. 11, 46–52 (2013).
8. F. A. Khalifa, N. A. Semary, H. M. El-Sayed, and M. M. Hadhoud, "Local detectors and descriptors for object class recognition," Int. J. of Intelligent Systems and Applications, Vol. 7, No. 10, 12–18 (2015).
9. A. Ziomek and M. Oszust, "Evaluation of interest point detectors in presence of noise," Int. J. Intelligent Systems and Applications, Vol. 8, No. 3, 26–33 (2016).
10. S. Fortune, "Voronoi diagrams and Delaunay triangulations," in: Handbook of Discrete and Computational Geometry, Chap. 27, 3rd Edition, CRC Press, Boca Raton, USA (2017), pp. 705–721.
11. S. Meiser, "Point location in arrangements of hyperplanes," Inform. and Comput., Vol. 106, No. 2, 286–303 (1993).
12. A. Andoni and P. Indyk, "Nearest neighbors in high-dimensional spaces," in: Handbook of Discrete and Computational Geometry, Chap. 43, 3rd Edition, CRC Press, Boca Raton, USA (2017), pp. 1133–1153.
13. R. Weber, H. Schek, and S. Blott, "A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces," in: Proc. VLDB'98 (1998), pp. 194–205.
14. S. Arya, D. Mount, N. Netanyahu, R. Silverman, and A. Wu, "An optimal algorithm for approximate nearest neighbor searching fixed dimensions," Journal of the ACM, Vol. 45, No. 6, 891–923 (1998).
15. A. Andoni and P. Indyk, "Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions," Communications of the ACM, Vol. 51. No. 1, 117–122 (2008).
16. S. Har-Peled, P. Indyk, and R. Motwani, "Approximate nearest neighbor: Towards removing the curse of dimensionality," Theory Comput., Vol. 8, 321–350 (2012).
17. D. M. W. Powers, "Evaluation: From precision, recall and F-measure to ROC, informedness, markedness and correlation," J. of Machine Learning Tech., Vol. 2, No. 1, 37–63 (2011).
18. R. Das, S. Thepade, and S. Ghosh, "Content based image recognition by information fusion with multiview features. I," J. Information Technology and Computer Science, Vol. 7, No. 10, 61–73 (2015).
19. S. Ramaswamy and K. Rose, "Adaptive cluster distance bounding for high-dimensional indexing," IEEE Trans. on KDE, Vol. 23, No. 6, 815–830 (2011).
20. M. Muja and D. G. Lowe, "Scalable nearest neighbor algorithms for high dimensional data," IEEE TPAMI, Vol. 36, No. 11, 2227–2240 (2014).
21. A. Shrivastava and P. Li, "Asymmetric minwise hashing for indexing binary inner products and set containment," in: Proc. WWW'15 (2015), pp. 981–991.
22. M. Charikar, "Similarity estimation techniques from rounding algorithms," in: Proc. STOC'02 (2002), pp. 380–388.
23. M. Aumuller, T. Christiani, R. Pagh, and F. Silvestr, Distance Sensitive Hashing. arXiv:1703.07867. 22 Mar 2017.
24. A. Andoni, M. Datar, N. Immorlica, P. Indyk, and V. S. Mirrokni, "Locality-sensitive hashing using stable distributions," in: Nearest Neighbor Methods for Learning and Vision: Theory and Practice, MIT Press, Cambridge (2006), pp. 61–72.
25. N. Pham, "Hybrid LSH: Faster near neighbors reporting in high-dimensional space," in: Proc. EDBT'17 (2017), pp. 454–457.
26. J. Wang, H. T. Shen, J. Song, and J. Ji, Hashing for Similarity Search: A Survey. arXiv:1408.2927. 13 Aug 2014.
27. J. Tang and Y. Tian, "A systematic review on minwise hashing algorithms," Annals of Data Science, Vol. 3, No. 4, 445–468 (2016).
28. B. Kulis and K. Grauman, "Kernelized locality-sensitive hashing," IEEE Trans. PAMI, Vol. 34, No. 6, 1092–1104 (2012).
29. Y. Mu and S. Yan, "Non-metric locality sensitive hashing," in: Proc. AAAI'10 (2010), pp. 539–544.

30. A. Andoni and P. Indyk, “Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions,” in: Proc. FOCS’06 (2006), pp. 459-468.
31. H. Jegou, L. Amsaleg, C. Schmid, and P. Gros, “Query-adaptive locality sensitive hashing,” in: Proc. ICASSP’08 (2008), pp. 825–828.
32. F. Chierichetti and R. Kumar, “Lsh-preserving functions and their applications,” J. ACM, Vol. 62, No. 5, 33:1–33:25 (2015).
33. F. Chierichetti, R. Kumar, A. Panconesi, and E. Terolli, “The distortion of locality sensitive hashing,” in: Proc. ITCS’17 (2017), p. 23.
34. A. Sokolov, “Investigation of accelerated search for close text sequences with the help of vector representations,” Cybernetics and Systems Analysis, Vol. 44, No. 4, 493–506 (2008).
35. A. Andoni, R. Krauthgamer, and I. P. Razenshteyn, “Sketching and embedding are equivalent for norms,” in: Proc. STOC’15 (2015), pp. 479–488.
36. A. Andoni, P. Indyk, T. Laarhoven, I. Razenshteyn, and L. Schmidt, “Practical and optimal LSH for angular distance,” in: Proc. NIPS’15 (2015), pp. 1225-1233.
37. K. Terasawa and Y. Tanaka, “Spherical lsh for approximate nearest neighbor search on unit hypersphere,” in: Proc. WADS’07 (2007), pp. 27–38.
38. K. Eshghi and S. Rajaram, “Locality sensitive hash functions based on concomitant rank order statistics,” in: Proc. KDD’08 (2008), pp. 221–229.
39. A. Andoni and I. Razenshteyn, “Optimal data-dependent hashing for approximate near neighbors,” in: Proc. STOC’15 (2015), pp. 793–801.
40. T. Laarhoven, “Hypercube LSH for approximate near neighbors,” in: Proc. MFCS’17 (2017).
41. C. Kennedy and R. Ward, “Fast cross-polytope locality-sensitive hashing,” in: Proc. ITCS’17 (2017).
42. R. Panigrahy, “Entropy based nearest neighbor search in high dimensions,” in: Proc. SODA’06 (2006), pp. 1186–1195.
43. Q. Lv, W. Josephson, Z. Wang, M. Charikar, and K. Li, “Multi-probe lsh: efficient indexing for high-dimensional similarity search,” in: Proc. VLDB’07 (2007), pp. 950–961.
44. A. Joly and O. Buisson, “A posteriori multi-probe locality sensitive hashing,” in: Proc. MM’08 (2008), pp. 209–218.
45. M. Slaney, Y. Lifshits, and J. He, “Optimal parameters for locality-sensitive hashing,” Proc. IEEE, Vol. 100, No. 9, 2604–2623 (2012).
46. M. Kapralov, “Smooth tradeoffs between insert and query complexity in nearest neighbor search,” in: Proc. PODS’15 (2015), pp. 329–342.
47. T. D. Ahle, M. Aumuller, and R. Pagh, “Parameter-free locality sensitive hashing for spherical range reporting,” in: Proc. SODA’17 (2017), pp. 239–256.
48. A. Pacuk, P. Sankowski, K. Wegrzycki, and P. Wygocki, “Locality-sensitive hashing without false negatives for lp,” in: Proc. COCOON’16 (2016), pp. 105–118.
49. P. Wygocki, On Fast Bounded Locality Sensitive Hashing. arXiv:1704.05902. 19 Apr 2017.
50. W. Dong, Z. Wang, W. Josephson, M. Charikar, and K. Li, “Modeling lsh for performance tuning,” in: Proc. CIKM’08 (2008), pp. 669–678.
51. P. Flajolet, E. Fusy, O. Gandouet, and F. Meunier, “Hyperloglog: The analysis of a near-optimal cardinality estimation algorithm,” in: Proc. AofA’07 (2007), pp. 127–146.
52. A. Chakrabarti, V. Satuluri, A. Srivathsan, and S. Parthasarathy, “A Bayesian perspective on locality sensitive hashing with extensions for kernel methods,” ACM TKDD, Vol. 10, No. 2, 19:1–19:32 (2015).
53. M. Bawa, T. Condie, and P. Ganesan, “Lsh forest: Self-tuning indexes for similarity search,” in: Proc. WWW’05 (2005), pp. 651–660.
54. A. Andoni, I. Razenshteyn, and N. Shekel Nosatzki, “Lsh forest: Practical algorithms made theoretical,” in: Proc. SODA’17 (2017), pp. 67–78.
55. Y. Tao, K. Yi, C. Sheng, and P. Kalnis, “Efficient and accurate nearest neighbor and closest pair search in high dimensional space,” ACM TODS, Vol. 35, No. 3, 20:1–20:46 (2010).
56. J. K. Lawder and P. J. H. King, “Querying multi-dimensional data indexed using the Hilbert space filling curve,” ACM SIGMOD Record, Vol. 30, No. 1, 19–24 (2001).
57. D. Comer, “The ubiquitous B-tree,” ACM Comput. Surv., Vol. 11, 121–138 (1979).
58. Y. Liu, J. Cui, Z. Huang, H. Li, and H. T. Shen, “Sk-lsh: An efficient index structure for approximate nearest neighbor search,” in: Proc. VLDB Endowment, Vol. 7, No. 9, 745-756 (2014).
59. J. Chen, C. He, G. Hu, and J. Shao, “SELSH: A hashing scheme for approximate similarity search with early stop condition,” in: Proc. MMM’16, Vol. 2 (2016), pp. 104–115.
60. F. Hao, J. Daugman, and P. Zielinski, “A fast search algorithm for a large fuzzy database,” IEEE Trans. Information Forensics and Security, Vol. 3, No. 2, 203–212 (2008).

61. K. Ling and G. Wu, “Frequency based locality sensitive hashing,” in: Proc. ICMT’11 (2011), pp. 4929–4932.
62. J. Gan, J. Feng, Q. Fang, and W. Ng, “Locality-sensitive hashing scheme based on dynamic collision counting,” in: Proc. SIGMOD’12 (2012), pp. 541–552.
63. Y. Zheng, Q. Guo, A. K. H. Tung, and S. Wu, “LazyLSH: Approximate nearest neighbor search for multiple distance functions with a single index,” in Proc. SIGMOD’16 (2016), pp. 2023–2037.
64. Q. Huang, J. Feng, Y. Zhang, Q. Fang, and W. Ng, “Query-aware locality-sensitive hashing for approximate nearest neighbor search,” Proc. VLDB Endowment, Vol 9, No. 1, 1–12 (2015).
65. X. Zhang, M. Wang, and J. Cui, “Efficient indexing of binary LSH for high dimensional nearest neighbor,” Neurocomputing, Vol. 213, 24–33 (2016).
66. M. Norouzi, A. Punjani, and D. J. Fleet, “Fast exact search in Hamming space with multi-index hashing,” IEEE Trans. PAMI, Vol. 36, No. 6, 1107–1119 (2014).
67. J. Gao, H. V. Jagadish, B. C. Ooi, and S. Wang, “Selective hashing: Closing the gap between radius search and k-NN search,” in: Proc. SIGKDD’15 (2015), pp. 349–358.
68. A. Andoni, T. Laarhoven, I. Razenshteyn, and E. Waingarten, “Optimal hashing-based time-space trade-offs for approximate near neighbors” in: Proc. SODA’17 (2017), pp. 47–66.
69. A. Becker, L. Ducas, N. Gama, and T. Laarhoven, “New directions in nearest neighbor searching with applications to lattice sieving,” in: Proc. SODA’16 (2016), pp. 10–24.
70. T. Christiani, “A framework for similarity search with space-time tradeoffs using locality-sensitive filtering,” in: Proc. SODA’17 (2017), pp. 31–46.
71. D. A. Rachkovskij, I. S. Misuno, and S. V. Slipchenko, “Randomized projective methods for construction of binary sparse vector representations,” Cybernetics and Systems Analysis, Vol. 48, No. 1, 146–156 (2012).
72. D. A. Rachkovskij, “Formation of similarity-reflecting binary vectors with random binary projections,” Cybernetics and Systems Analysis, Vol. 51, No. 2, 313–323 (2015).
73. R. Donaldson, A. Gupta, Y. Plan, and T. Reimer, Random Mappings Designed for Commercial Search Engines. arXiv:1507.05929. 21 Jul 2015.
74. S. Ferdowsi, S. Voloshynovskiy, D. Kostadinov, and T. Holotyak, “Fast content identification in highdimensional feature spaces using sparse ternary codes,” in: Proc. WIFS’16 (2016), pp. 1–6.
75. G. Valiant, “Finding correlations in subquadratic time, with applications to learning parities and the closest pair problem,” J. ACM, Vol. 62, No. 2, 13:1–13:45 (2015).
76. H. L. Nguyen, Algorithms for High Dimensional Data, PhD Thesis, Princeton University (2014). URL: http://arks.princeton.edu/ark:/88435/dsp01b8515q61f.
77. A. Rahimi and B. Recht, “Random features for large-scale kernel machine,” in: Proc. NIPS’07 (2007), pp. 1177–1184.
78. R. O’Donnell, Y. Wu, and Y. Zhou, “Optimal lower bounds for locality sensitive hashing (except when q is tiny),” ACM TOCS, Vol. 6, No. 1, 5.1–5.13 (2014).
79. J. Wang, W. Liu, S. Kumar, and S.-F. Chang, “Learning to hash for indexing big data: A survey,” Proc. IEEE, Vol. 104, No. 1, 34–57 (2016).
80. J. Wang, T. Zhang, J. Song, N. Sebe, and H. T. Shen, “A Survey on Learning to Hash,” IEEE Trans. PAMI. doi:10.1109/TPAMI.2017.2699960.
81. L. Gao, J. Song, X. Liu, J. Shao, J. Liu, and J. Shao, “Learning in high-dimensional multimedia data: The state of the art,” Multimedia Systems, Vol. 23, No. 3, 303–313 (2017).
82. W. Mou and L. Wang, “A refined analysis of lsh for well-dispersed data points,” in: Proc. ANALCO’17 (2017), pp. 174–182.
83. A. Andoni, P. Indyk, H. L. Nguyen, and I. Razenshteyn, “Beyond locality-sensitive hashing,” in: Proc. SODA’14 (2014), pp. 1018–1028.
84. A. Andoni and I. Razenshteyn, “Tight lower bounds for data-dependent locality-sensitive hashing,” in: Proc. SoCG’16 (2016), pp. 9:1–9:11.
85. V. I. Gritsenko, D. A. Rachkovskij, A. A. Frolov, R. Gayler, D. Kleyko, and E. Osipov, “Neural distributed autoassociative memories: A survey,” Cybernetics and Computer Engineering, No. 2 (188), 5–35 (2017).
86. Y. Wang, A. Shrivastava, and J. Ryu, FLASH: Randomized Algorithms Accelerated over CPU-GPU for Ultra-High Dimensional Similarity Search. arXiv:1709.01190. 4 Sep 2017.
87. A. Shrivastava, “Optimal densification for fast and accurate minwise hashing,” in: Proc. ICML’17 (2017), pp. 3154–3163.