Large-Scale Similarity Joins With Guarantees

Rasmus Pagh[*1]

1 **IT University of Copenhagen, Denmark**
pagh@itu.dk

Abstract

The ability to handle noisy or imprecise data is becoming increasingly important in computing. In the database community the notion of similarity join has been studied extensively, yet existing solutions have offered weak performance guarantees. Either they are based on deterministic filtering techniques that often, but not always, succeed in reducing computational costs, or they are based on randomized techniques that have improved guarantees on computational cost but come with a probability of not returning the correct result. The aim of this paper is to give an overview of randomized techniques for high-dimensional similarity search, and discuss recent advances towards making these techniques more widely applicable by eliminating probability of error and improving the locality of data access.

If I am not me, then who the hell am I?
Douglas Quaid (Arnold Schwarzenegger) in *Total Recall*.

1 Introduction

In their famous book *Perceptrons* [1] from 1969, Minsky and Papert formulated a simple indexing problem: "Store a set S of D-dimensional binary vectors, such that for a query vector q, and a tolerance r, we can quickly answer if there exists a vector in S that differs from q in at most r positions." To this day the achievable space/time trade-offs for this indexing problem remain unknown, and even the best solutions offer only weak performance guarantees for large r and D compared to the guarantees that are known for exact or 1-dimensional search (hash tables, B-trees). At the same time "tolerance" in data processing and analysis is gaining importance, as systems are increasingly dealing with imprecise data such as: different textual representations of the same object (e.g., in data reconciliation), slightly modified versions of a string (e.g., in plagiarism detection), or feature vectors whose similarity tells us about the affinity of two objects (e.g., in recommender systems).

The reporting version of Minsky and Papert's problem, where the task is to search for all vectors in S within distance r from q, is usually referred to as the *near neighbor* problem, and has been considered as an approach to tolerant indexing for many different spaces and metrics. In database settings it will often be the case that there is a *set* Q of vectors[1] for which we want to find all near neighbors in S. If we let $d(q, x)$ denote the distance between vectors q and x, the *similarity join* of Q and S with tolerance r is the set

$$Q \bowtie_r S = \{(q, x) \in Q \times S \mid d(q, x) \leq r\} \ .$$

The term "similarity join" suggests that we join vectors that are similar, defined as having distance smaller than r. Sometimes it is more natural to work with a measure of similarity

* Supported by the European Research Council under the European Union's 7th Framework Programme (FP7/2007-2013) / ERC grant agreement no. 614331.

[1] While the near neighbor problem can be defined for arbitrary spaces we will refer to elements of the space as "vectors".

18th International Conference on Database Theory (ICDT'15).
Editors: Marcelo Arenas and Martin Ugarte; pp. 1–10

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

rather than distance, for example *cosine* or *Jaccard* similarity. This does not fundamentally change the problem since for each similarity measure we can define a distance measure that is a decreasing function of similarity.

Some applications do not specify a fixed search radius r, but rather ask to find the k closest vectors in S for each $q \in Q$. This variant is referred to as a *k-nearest-neighbor* (kNN) similarity join. Here we do not consider kNN similarity joins, but note that there are asymptotically efficient reductions for answering kNN queries using near neighbor queries [2].

While our focus is on theoretical aspects of computing similarity joins, we note that the problem (and its kNN variation) is of broad practical importance and has been a subject of extensive study in recent years in several application-oriented fields:

- databases (e.g. [3–13]),
- information retrieval (e.g. [14–16]), and
- knowledge discovery (e.g. [17–21]).

Scope of this paper. The aim of this paper is to give an overview of the main techniques used for high-dimensional similarity search, and discuss recent advances towards making these techniques more widely applicable by eliminating probability of error and improving the locality of data access.

We focus on *randomized* techniques that scale well to *high-dimensional* data. (In low dimensions there exist efficient indexing methods for similarity search, and these translate into efficient similarity join methods.) For similarity joins it is generally better to take advantage of the fact that many queries have to be answered, and avoid creating a standard index, so we omit discussion of the extensive body of work dealing with *indexes* for similarity search.

We refer to the recent book by Augsten and Böhlen [22] for a survey of a broader spectrum of algorithms for similarity join than we are able to cover here.

2 Techniques for high-dimensional similarity joins

To focus on basic techniques we mainly consider the simple case of D-dimensional binary vectors and Hamming distances. Also, we assume for simplicity that the binary vectors are explicitly stored, but binary vectors with Hamming weight much smaller than D could advantageously be stored using a sparse representation.

2.1 Approximation

Following the seminal papers of Gionis, Har-Peled, Indyk, and Motwani [2, 25] a large literature has focused on *approximate* similarity join algorithms that may have *false negatives*, i.e., pairs that have distance at most r but fail to be reported by the algorithm. In this setting it is of course desirable to achieve the highest possible *recall*, defined as the fraction of the join result returned by the algorithm.

The possibility of false negatives should not be confused with *over*-approximation where the join result is likely to contain "false positive" pairs at distance up to cr, for some approximation factor c. False positives can be eliminated by checking the distances of all pairs, which is relatively cheap if the *precision* is not too low, i.e., when there is not a large fraction of such false positives. It has been observed that many real-life data sets have low intrinsic dimension in the sense that the number of vectors within distance cr is within a reasonably small factor of the number of vectors within distance r when c is a small constant.

2.2 Candidate set generation

Many approximate similarity join algorithms work by first generating a set of *candidate pairs* $C \subseteq Q \times S$. For each $(q, x) \in C$ the distance $d(q, x)$ is then either computed exactly or estimated with sufficient precision to determine whether or not $d(q, x) \leq r$. More generally one can think about a gradual process where a distance estimate is refined for each pair until it is possible to determine whether $d(q, x) \leq r$. Many results on similarity join can be seen as methods for computing good estimates of distances based on a small amount of information about vectors. Lower bounds from communication complexity (see e.g. [23] and its references) tell us that it is in general not possible to efficiently and accurately estimate distances between two vectors based on a small amount of information about each. But in many settings even crude estimates can give rise to substantial savings compared to a naïve solution that computes all distances precisely. For example, the Hamming distance between vectors q and x always lies between $|||q|| - ||x|||$ and $||q|| + ||x||$, where $||\cdot||$ denotes the Hamming weight, i.e., number of 1s.

Subquadratic algorithms. Ideally, to be able to scale to large set sizes we would like to not spend time on every pair in $Q \times S$. For simplicity of discussion suppose that $|Q| = |S| = n$ such that $|Q \times S| = n^2$. Of course, if the join result has size close to n^2 we cannot hope to use subquadratic time. But for large n it will normally only be of interest to compute $Q \bowtie_r S$ when $|Q \bowtie_r S| \ll n^2$, so we focus on this case.

Some algorithms achieve subquadratic performance for certain data distributions. For example Bayardo et al. [14] exploit that cosine similarity of most pairs of vectors (in information retrieval data sets) is small unless they share high weight features. For data sets where many pairs do not share such high weight features the processing time can be substantially subquadratic. Another example comes from association mining in genetic data where Achlioptas et al. [17], building upon seminal work of Paturi et al. [24], obtained running time empirically proportional to $n^{3/2}$.

Curse of dimensionality. To obtain the widest possible applicability we would ideally like to guarantee subquadratic running time *regardless* of data distribution. It is commonly believed that such a general result is not possible due to the *curse of dimensionality*. Indeed, no similarity join algorithms with good time bounds in terms of parameters n and D are known. This means that, unless the general belief is wrong, we need to settle for results that take some aspect of the data distribution into account. Intuitively, the most difficult case occurs when all distances are close to r, since it is hard to determine whether or not to include a pair without computing its distance exactly. Most data sets have a *distribution of distances* that is much more well-behaved, at least up to a certain radius $r_{\max}$ above which the similarity join size explodes. Some of the strongest theoretical results that guarantee subquadratic performance (e.g. [2, 25]) rest on the assumption that we are considering r that is not too close to $r_{\max}$. In other words, there is not a huge number of pairs with distance slightly above r. Often the ratio $r_{\max}/r$ is thought of as a parameter c that specifies how well distances need to be approximated to distinguish pairs at distance r from pairs at distance $r_{\max}$.

2.3 Locality-sensitive hashing.

Paturi et al. [24] pioneered an approach to candidate set generation that results in subquadratic time for similarity join[2] as soon as the above-mentioned approximation ratio c is larger than 1.

[2] In fact, they considered a simplified setting where the task is to find a *single* pair at distance less than r in a binary data set that is otherwise random. But their technique extends to the more general setting of similarity join computation.

Their technique can be seems as an early instance of *locality sensitive hashing* (LSH), which creates a candidate set using a sequence of specially constructed hash functions $h_1, h_2, \ldots, h_t$ chosen from a family of functions where the collision probability $\Pr[h_i(q) = h_i(x)]$ is a non-increasing function of the distance $d(q, x)$. For each hash function h_i, every pair in $Q \times S$ with colliding hash values is included as a candidate pair, i.e.:

$$C = \bigcup_{i=1}^{t} \{(q, x) \in Q \times S \mid h_i(q) = h_i(x)\} \ . \tag{1}$$

Computing C can be seen as a sequence of t equi-joins with hash value as join attribute. If we are not concerned with the possibility of generating a candidate pair more than once, these joins can be processed independently, and the space required for each join is linear in the input size. While these joins can of course individually be efficiently computed, the number of joins needed can be large, so it is not obvious that computing them one by one is the best solution. Indeed, in Section 4 we give an overview of recent work highlighting how other approaches can give rise to better temporal locality of memory accesses, resulting in more efficient algorithms in memory hierarchies.

LSH construction. For binary vectors and Hamming distances, the currently best LSH construction (from [24, 25]) is remarkably simple. It works by selecting (or sampling) a random set of bits from the vector, i.e., for some randomly chosen $a_i \in \{0, 1\}^D$ we have:

$$h_i(x) = x \wedge a_i, \tag{2}$$

where $\wedge$ denotes bitwise conjunction, i.e., the `&` operator in many programming languages. The Hamming weight of a_i is chosen to ensure "small" collision probability for the large majority of pairs in $Q \times S$ that have distance above cr, for some c. If the aim is to minimize the total number of vector comparisons plus hash function evaluations the best choice of a_i (for large D) is a random vector with $\|a_i\| \approx D \ln(n)/(cr)$. This results in $\Pr[h_i(q) = h_i(x)] < 1/n$ when $d(q, x) > cr$, meaning that for each h_i there will in expectation be $\mathcal{O}(n)$ candidate pairs that have distance above cr. A common way to think about this is as a *filter* that is able to remove all but a few of the up to n^2 pairs in $Q \times S$ that are not "close to being in the join result". If we have $d(q, x) = r$ the same choice of a_i yields $\Pr[h_i(q) = h_i(x)] \approx n^{-1/c}$. So for each hash function h_i the probability that (q, x) is added to C is quite small. This can addressed by using $t > n^{1/c}$ hash functions, which for an independent sequence of hash functions reduces the probability that $(q, x) \notin C$ (such that the pair becomes a false negative) to $(1 - n^{-1/c})^t < 1/e$. Further increasing the parameter t reduces, but does not eliminate, the probability of false negatives.

We note that LSH families for many other distance (and similarity) measures have been developed. The 2012 Paris Kanellakis Theory And Practice Award went to Andrei Broder, Moses Charikar, and Piotr Indyk for their contributions to work on locality sensitive hashing for fundamental measures, e.g. [25–28]. LSH methods are by definition randomized and have often been seen as inherently approximate in the sense that they must have a nonzero false negative probability. However, as we will see in Section 3 there are randomized filtering techniques that guarantee *total recall*, i.e., no false negatives. As such, the borderline between LSH techniques and other filtering technique is blurred, or even meaningless.

2.4 Aggregation

In some cases it is possible to gather information on a *set* of vectors that allows filtering of pairs involving any element of the set. One such technique is based on *aggregation*. As an example, Valiant [29] showed that if we consider $Q' \subset Q$ and $S' \subset S$ there are situations in which we can show emptiness of $Q' \bowtie_r S'$ based only on aggregate vectors $\sigma_{Q'} = \sum_{q \in Q'} (2q - 1)s(q)$ and

$\sigma_{S'} = \sum_{x \in S'} (2x-1)s(x)$, where $s : \{0,1\}^D \to \{-1,+1\}$ is a random function and the arithmetic is over the integers. We consider the dot product

$$\sigma_{Q'} \cdot \sigma_{S'} = \sum_{q \in Q'} \sum_{x \in S'} (s(q)(2q - 1^D)) \cdot (s(x)(2x - 1^D)) \ . \tag{3}$$

The value of (3) is useful for example if the dot product $(2q - 1^D) \cdot (2x - 1^D)$ is close to 0 for all but at most one pair $(q, x) \in Q \times S$. Since $(2q - 1^D) \cdot (2x - 1^D) = D - 2d(q, x)$ a pair with small Hamming distance has a contribution to (3) of either close to $-D$ or close to $+D$. This means that the presence of a close pair can be detected (with a certain probability of error) by considering whether $|\sigma_{Q'} \cdot \sigma_{S'}|$ is close to 0 or close to D. In a theoretical breakthrough [29], Valiant showed how to combine this technique with fast matrix multiplication (for computing all dot products of aggregates) to achieve sub-quadratic running time even in cases where most distances are very close to r. It remains to be seen if aggregation techniques can yield algorithms that are good in both theory and practice. Promising experimental work on a different aggregation technique was carried out by Low and Zheng [30], independently of Valiant, in the context of indexing.

3 Addressing the issue of false negatives

We now consider ways of eliminating false negatives, or in other words achieving total recall. As a starting observation, note that the non-zero probability that q and x do not collide under any h_i, as defined in (2), is *not* due to the fact that each a_i is random. Indeed, if we choose $a_1, \ldots, a_t$ as a random permutation of the set of all vectors of Hamming weight $D - r$ we will be guaranteed that $h_i(q) = h_i(x)$ for some i, while vectors at distance above r will have no collisions. We say that such a sequence of vectors is *covering* for distance r. The reason that traditional LSH constructions are not covering is that the hash functions are chosen *independently*, which inherently means that there will be a probability of error. Though the example above is extreme — it requires t to be very large — it does show that suitably *correlating* the hash functions in the sequence can ensure that all pairs within distance r are included in the candidate set.

More efficient constructions. Following an early work of Greene et al. [31], Arasu et al. [32] presented a much more efficient construction of a covering vector sequence. Their first observation is that if $t = r + 1$ and we choose $a_1, \ldots, a_t$ that are disjoint ($a_i \wedge a_j = 0^D$ for all $i \neq j$) and covering all dimensions ($\bigvee_i a_i = 1^D$) then vectors at distance at most r will be identical in the bits indicated by at least one vector a_i, such that $h_i(q) = h_i(x)$.

While this approach ensures that we generate every pair in $Q \bowtie_r S$ as a candidate, it is not clear how well this sequence of functions is able to filter away pairs at distances larger than r from the candidate set. Norouzi et al. [33] recently considered, independently of Arasu et al. [32], the special case where the binary vectors are random. But for general (worst case) data it could be that all differences between q and x appear in the positions indicated by a_1, in which case there will be hash collisions for all functions except h_1. To address this, Arasu et al. propose to initially apply a random permutation $\pi : \{1, \ldots, D\} \to \{1, \ldots, D\}$ to the dimensions. Abusing notation we let $\pi(x)$ denote the vector where $\pi(x)_i = x_{\pi(i)}$, so that the hash functions considered are defined by:

$$h'_i(x) = \pi(x) \wedge a_i \ . \tag{4}$$

As a practical consideration it may be advantageous to instead compute $\pi^{-1}(h'(x)) = x \wedge \pi^{-1}(a_i)$, since we can precompute the vectors $\pi^{-1}(a_i)$, and this change preserves collisions.

Analysis. We claim that including the permutation makes $\Pr\left[h'_i(q) = h'_i(x)\right]$, where the probability is over the choice of random permutation, depend only on the Hamming weight $\|a_i\|$

and the distance $d(q,x)$. Essentially, $h'_i(x)$ is sampling a set of $\|a_i\|$ positions from $\{1,\ldots,D\}$, without replacement, and we have a collision if none of these positions differ:

$$\Pr\left[h'_i(q) = h'_i(x)\right] = \binom{D - d(q,x)}{\|a_i\|} \Big/ \binom{D}{\|a_i\|} \ . \tag{5}$$

When D is large (and $D \gg d(q,x)$) this probability is well approximated by the probability when sampling with replacement, i.e.,

$$\Pr\left[h'_i(q) = h'_i(x)\right] \approx (1 - d(q,x)/D)^{\|a_i\|} \approx \exp(-\|a_i\|\, d(q,x)/D) \ . \tag{6}$$

As can be seen the collision probability decreases exponentially with $\|a_i\|$, so increasing the Hamming weights of $a_1,\ldots,a_t$ will increase filtering efficiency. Arasu et al. present a method for getting a covering sequence $a^*_1,\ldots,a^*_t$ of larger Hamming weight (no longer satisfying the disjointness condition). We refer to their paper for a precise description, and simply use $h^*_1,\ldots,h^*_t$ to denote the corresponding hash functions. Arasu et al. do not provide a general analysis, but it appears that the Hamming weight that their method can achieve with t functions is asymptotically (for large t and r) around $D\log_2(t)/(4r)$. This corresponds to a collision probability for each hash function h^*_i of roughly:

$$\Pr\left[h^*_i(q) = h^*_i(x)\right] \approx \exp(-\log_2(t) d(q,x)/(4r)) \approx t^{-0.36 d(q,x)/r} \ . \tag{7}$$

The probability that (q,x) is included in the candidate set can be upper bounded by a union bound over all t functions, as $t^{1-0.36\, d(q,x)/r}$. This means that for $d(q,x) > 3r$ we are likely not to include (q,x) as a candidate, and the probability that it becomes a candidate decreases polynomially with t. Just as for standard LSH, the best choice of t balances the cost of false positives in the candidate set with the cost of identifying candidate pairs more than once.

Discussion of optimality. If we consider (q,x) with $d(q,x) = r$ any covering sequence of vectors needs to ensure a collision, which requires, at least, that the expected number of collisions is 1. So if we use hash functions $h_1,\ldots,h_t$ then we must have $\sum_{i=1}^{t} \Pr\left[h_i(q) = h_i(x) \mid d(q,x) = r\right] \geq 1$. If we assume that there is a non-increasing function of $f : \mathbf{R} \to \mathbf{R}$ such that for each i, $\Pr\left[h_i(q) = h_i(x) \mid d(q,x) = cr\right] = f(c)$, then:

$$t \geq 1/f(1) \ . \tag{8}$$

According to (7) the construction of Arasu et al. [32] obtains $f(1) \approx t^{-0.36}$, i.e., $t \approx (1/f(1))^{2.77}$. If it is possible to obtain a sequence of hash functions of length close to the lower bound (8) it will be possible to match the performance of the classical LSH method of Gionis et al. [25]. A nonconstructive argument by the probabilistic method, using a sequence of randomly chosen vectors a_i, shows that $t = \mathcal{O}\left(r\log(D)/f(1)\right)$ suffices to ensure the existence of a covering sequence of vectors. However, this fact is of little help since we have no effective way of constructing such a sequence, or even checking its correctness.

4 Addressing the issue of data locality

An issue with the candidate generation method that we have outlined is that it makes poor use of the memory hierarchy. If the data set does not fit in internal memory it is likely that we will need to read every vector from external memory once for each hash function. In contrast, methods that explicitly consider all pairs of vectors can be implemented with good data locality by handling subsets of Q and S that fit in fast memory.

Bahmani et al. [18] suggested to use two levels of LSH to compute similarity joins more efficiently in a distributed setting, where the goal is to distribute the data such that most computations can be done separately on each node. Roughly, the top-level LSH distributes the data

to nodes such that it is reasonably evenly distributed, and the second-level LSH improves the filtering to decrease the computational cost.

In a recent paper [34] we take this idea further and analyze the resulting algorithm in the *I/O model* (see e.g. [35]), where the aim is to minimize the number of block transfers between slow and fast memory. Consider a setting where standard LSH-based similarity joins use n^ρ hash functions, for a parameter $\rho \in (0;1)$. If n significantly exceeds the number M of vectors that fit in fast memory, this means that the join algorithm will need to transfer each element n^ρ times between fast and slow memory. This is not always better than a naïve block nested loop join algorithm, which transfers each vector n/M times. Our main finding is that both of these bounds can be improved for large n, to $\mathcal{O}\left((n/M)^\rho \operatorname{poly}\log n\right)$ transfers of each vector, under a reasonable assumption on the distance distribution (essentially that the number of pairs in $Q \times S$ at distance close to r is not huge).

A recursive similarity join algorithm. The algorithm described in our paper [34] is designed with ease of analysis in mind, so it may not be fully practical — in particular due to the $\operatorname{poly}\log n$ factor. Here we will consider a simple variant that highlights the main idea of [34] and is potentially useful. The idea is to use a shorter sequence of LSH functions $h_1, \ldots, h_L$ that is just powerful enough to filter away a *constant fraction* of the candidate pairs in $Q \times S$. This step alone will leave many false positives, but it can be applied *recursively* to each bucket of each hash function to increase filtering efficiency. We will soon describe the base case of the recursion, but first state the general case, where $C(Q,S)$ denotes our recursive function for computing a candidate set. Let V denote the set of possible hash values; then:

$$C(Q,S) = \bigcup_{i=1}^{L} \bigcup_{v \in V} C(\{q \in Q \mid h_i(q) = v\}, \{x \in S \mid h_i(x) = v\}), \tag{9}$$

where the sequence of hash functions is chosen randomly and independently. Since a pair may collide more than once the candidate set should be thought of as a *multiset*, and the number of candidates proportional to the size of this multiset. The recursion in (9) effectively replaces the naïve candidate set $Q \times S$ by a multiset of colliding pairs of size:

$$\operatorname{coll}(Q,S) = \sum_{i=1}^{L} \sum_{v \in V} |\{q \in Q \mid h_i(q) = v\} \times \{x \in S \mid h_i(x) = v\}| \ . \tag{10}$$

We would like to apply (9) when it at least halves the number $\operatorname{coll}(Q,S)$ of candidate pairs compared to $|Q \times S|$. That is, as a base case we take:

$$C(Q,S) = Q \times S \text{ if } |Q \times S| \leq 2\operatorname{coll}(Q,S) \ . \tag{11}$$

Discussion. If we consider the sets $Q' \subseteq Q$ and $S' \subseteq S$ in a particular base case, all vectors will have identical values for a sequence of hash functions. This is in some ways similar to the common technique of composing several LSH functions to increase filtering efficiency. The essential difference is that many subproblems can share each hash function, so much of the work on distributing vectors into subproblems is shared. Since applying the LSH family to the base case fails to decrease the number of candidate pairs substantially, it must be the case that the majority of pairs in $Q' \times S'$ have distance not much above r, so the complexity of checking candidates can be bounded in terms of the number of pairs that have distance not much above r.

We outline the main idea in the analysis from our paper [34] as it applies to the simplified algorithm. If we restrict attention to a particular pair $(q,x) \in Q \times S$, the recursion can be seen as a *branching process* where the presence of (q,x) in a subproblem may lead to the pair being repeated in one or more recursive subproblems. The pair is ultimately considered if and only if it "survives" on some path in the recursion tree, all the way down to a base case (where all pairs

are considered as candidates). We would like to ensure that if $d(q, x) \leq r$ then (q, x) survives with certainty (or with good probability, if false negatives can be tolerated), while if $d(q, x) > cr$ for $c > 1$ the probability that (q, x) becomes a candidate decreases rapidly with c. The theory of branching processes (see e.g. [36]) says that this will be the case if the expected number of recursive calls involving (q, x) is at least 1 for $d(q, x) \leq r$ and less than 1 for $d(q, x) > cr$.

An advantage of a recursive algorithm is that once subproblems are small enough to fit in fast memory they will produce no further transfers between fast and slow memory. When solving the base case we may also use a recursive approach by splitting $Q \times S$ into four candidate sets each involving half of Q and S.

We stress that an arbitrary sequence of hash functions can be used, including those described in Section 3. In this case, the similarity join algorithm will achieve total recall.

5 Conclusion and open problems

We have argued that randomized techniques related to those traditionally used in LSH are applicable to similarity joins in which false negatives do not occur. It remains to be seen to what extent such techniques can be extended to other spaces and metrics, such as Euclidean or Manhattan distances, but non-constructive arguments suggest that at least some nontrivial results are possible. Even in the case of binary vectors it is an interesting question if the construction of Arasu et al. [32] can be improved, and in that case how close to the lower bound (8) it is possible to get.

Finally, recent developments on indexes for approximate similarity search [37] suggest that it may be possible to further improve LSH-based similarity join algorithms.

Acknowledgement. The author thanks Ninh Pham, Francesco Silvestri, and Matthew Skala for useful feedback on a preliminary version of this paper.

References

1 M. L. Minsky and S. A. Papert, *Perceptrons - Expanded Edition: An Introduction to Computational Geometry*. MIT press, 1987.

2 S. Har-Peled, P. Indyk, and R. Motwani, "Approximate nearest neighbor: Towards removing the curse of dimensionality.," *Theory of computing*, vol. 8, no. 1, pp. 321–350, 2012.

3 S. Chaudhuri, V. Ganti, and R. Kaushik, "A primitive operator for similarity joins in data cleaning," in *Proceedings of ICDE*, p. 5, 2006.

4 Y. Chen and J. M. Patel, "Efficient evaluation of all-nearest-neighbor queries," in *Proceedings of ICDE*, pp. 1056–1065, IEEE, 2007.

5 E. Cohen, M. Datar, S. Fujiwara, A. Gionis, P. Indyk, R. Motwani, J. D. Ullman, and C. Yang, "Finding interesting associations without support pruning," *IEEE Trans. Knowl. Data Eng.*, vol. 13, no. 1, pp. 64–78, 2001.

6 E. H. Jacox and H. Samet, "Metric space similarity joins," *ACM Transactions on Database Systems (TODS)*, vol. 33, no. 2, p. 7, 2008.

7 Y. Jiang, D. Deng, J. Wang, G. Li, and J. Feng, "Efficient parallel partition-based algorithms for similarity search and join with edit distance constraints," in *Proceedings of Joint EDBT/ICDT Workshops*, pp. 341–348, ACM, 2013.

8 G. Li, D. Deng, J. Wang, and J. Feng, "Pass-join: A partition-based method for similarity joins," *Proceedings of the VLDB Endowment*, vol. 5, no. 3, pp. 253–264, 2011.

9 J. Lu, C. Lin, W. Wang, C. Li, and H. Wang, "String similarity measures and joins with synonyms," in *Proceedings of SIGMOD*, pp. 373–384, ACM, 2013.

10 Y. N. Silva, W. G. Aref, and M. H. Ali, "The similarity join database operator," in *Proceedings of ICDE*, pp. 892–903, IEEE, 2010.

11 J. Wang, G. Li, and J. Fe, "Fast-join: An efficient method for fuzzy token matching based string similarity join," in *Proceedings of ICDE*, pp. 458–469, IEEE, 2011.

12 J. Wang, G. Li, and J. Feng, "Can we beat the prefix filtering?: an adaptive framework for similarity join and search," in *Proceedings of SIGMOD*, pp. 85–96, ACM, 2012.

13 C. Xia, H. Lu, B. C. Ooi, and J. Hu, "Gorder: an efficient method for knn join processing," in *Proceedings of VLDB*, pp. 756–767, VLDB Endowment, 2004.

14 R. J. Bayardo, Y. Ma, and R. Srikant, "Scaling up all pairs similarity search," in *Proceedings of WWW*, pp. 131–140, 2007.

15 A. Das, M. Datar, A. Garg, and S. Rajaram, "Google news personalization: scalable online collaborative filtering," in *Proceedings of WWW*, pp. 271–280, 2007.

16 C. Xiao, W. Wang, X. Lin, and J. X. Yu, "Efficient similarity joins for near duplicate detection," in *Proceedings of WWW*, pp. 131–140, 2008.

17 P. Achlioptas, B. Schölkopf, and K. Borgwardt, "Two-locus association mapping in subquadratic time," in *Proceedings of KDD*, pp. 726–734, ACM, 2011.

18 B. Bahmani, A. Goel, and R. Shinde, "Efficient distributed locality sensitive hashing," in *Proceedings of CIKM*, pp. 2174–2178, 2012.

19 Y. Wang, A. Metwally, and S. Parthasarathy, "Scalable all-pairs similarity search in metric spaces," in *Proceedings of KDD*, pp. 829–837, 2013.

20 R. B. Zadeh and A. Goel, "Dimension independent similarity computation," *The Journal of Machine Learning Research*, vol. 14, no. 1, pp. 1605–1626, 2013.

21 X. Zhang, F. Zou, and W. Wang, "Fastanova: an efficient algorithm for genome-wide association study," in *Proceedings of KDD*, pp. 821–829, ACM, 2008.

22 N. Augsten and M. H. Böhlen, "Similarity joins in relational database systems," *Synthesis Lectures on Data Management*, vol. 5, no. 5, pp. 1–124, 2013.

23 R. Pagh, M. Stöckel, and D. P. Woodruff, "Is min-wise hashing optimal for summarizing set intersection?," in *Proceedings of PODS*, pp. 109–120, ACM, 2014.

24 R. Paturi, S. Rajasekaran, and J. Reif, "The Light Bulb Problem," *Information and Computation*, vol. 117, pp. 187–192, Mar. 1995.

25 A. Gionis, P. Indyk, and R. Motwani, "Similarity search in high dimensions via hashing," in *Proceedings of VLDB*, pp. 518–529, 1999.

26 A. Z. Broder, S. C. Glassman, M. S. Manasse, and G. Zweig, "Syntactic clustering of the web," *Computer Networks*, vol. 29, no. 8-13, pp. 1157–1166, 1997.

27 M. S. Charikar, "Similarity estimation techniques from rounding algorithms," in *Proceedings of STOC*, pp. 380–388, 2002.

28 M. Datar, N. Immorlica, P. Indyk, and V. S. Mirrokni, "Locality-sensitive hashing scheme based on p-stable distributions," in *Proceedings of SOCG*, pp. 253–262, 2004.

29 G. Valiant, "Finding Correlations in Subquadratic Time, with Applications to Learning Parities and Juntas," in *Proceedings of FOCS*, pp. 11–20, IEEE, Oct. 2012.

30 Y. Low and A. X. Zheng, "Fast top-k similarity queries via matrix compression," in *Proceedings of CIKM*, pp. 2070–2074, ACM, 2012.

31 D. Greene, M. Parnas, and F. Yao, "Multi-index hashing for information retrieval," in *Proceedings of FOCS*, pp. 722–731, IEEE, 1994.

32 A. Arasu, V. Ganti, and R. Kaushik, "Efficient exact set-similarity joins," in *Proceedings of VLDB*, pp. 918–929, 2006.

33 M. Norouzi, A. Punjani, and D. J. Fleet, "Fast search in hamming space with multi-index hashing," in *Proceedings of CVPR*, pp. 3108–3115, IEEE, 2012.

34 R. Pagh, N. Pham, F. Silvestri, and M. Stöckel, "I/O-efficient similarity join in high dimensions." Manuscript, 2015.

35 J. S. Vitter, *Algorithms and Data Structures for External Memory*. Now Publishers Inc., 2008.

36 T. E. Harris, *The theory of branching processes*. Courier Dover Publications, 2002.

37 A. Andoni and I. Razenshteyn, "Optimal data-dependent hashing for approximate near neighbors," *arXiv preprint arXiv:1501.01062*, 2015.