Disorder Inequality: A Combinatorial Approach to Nearest Neighbor Search

Navin Goyal
McGill University
navin@cs.mcgill.ca

Yury Lifshits
Steklov Institute of Mathematics at St.Petersburg
California Institute of Technology
yura@logic.pdmi.ras.ru

Hinrich Schütze
University of Stuttgart
hinrich@hotmail.com

ABSTRACT

We say that an algorithm for nearest neighbor search is combinatorial if only direct comparisons between two pairwise similarity values are allowed. Combinatorial algorithms for nearest neighbor search have two important advantages: (1) they do not map similarity values to artificial distance values and do not use triangle inequality for the latter, and (2) they work for arbitrarily complicated data representations and similarity functions.

In this paper we introduce a special property of the similarity function on a set S that leads to efficient combinatorial algorithms for S. Disorder constant $D(S)$ of a set S is defined to ensure the following inequality: if x is the a'th most similar object to z and y is the b'th most similar object to z, then x is among the $D(S) \cdot (a+b)$'th most similar objects to y.

Assuming that disorder is small we present the first two known combinatorial algorithms for nearest neighbors whose query time has logarithmic dependence on the size of S. The first one called Ranwalk is a randomized zero-error algorithm that always returns the exact nearest neighbor. It uses space quadratic in the input size in preprocessing, but is very efficient in query processing. The second algorithm, called Arwalk, uses near-linear space. It uses random choices in preprocessing, but the query processing is essentially deterministic. For every fixed query, there is only a small probability that the chosen data structure does not support it.

Finally, we show that for Reuters corpus average disorder is, indeed, quite small and that ranwalk efficiently computes the nearest neighbor in most cases.

Categories and Subject Descriptors

E.1 [**Data Structures**]; F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems: computations on discrete structures, sorting and searching; H.2.4 [**Database Management**]: Systems: query processing; H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval: retrieval models, search process; H.3.1 [**Information Storage and Retrieval**]: Content Analysis and Indexing: indexing methods; I.3.1 [**Pattern Recognition**]: Clustering: similarity measures

General Terms

Algorithms, Theory, Experimentation

Keywords

Nearest neighbor search, similarity search, proximity search, random walk, randomized algorithms, disorder inequality, disorder dimension, disorder constant

1. INTRODUCTION

Challenge.

Nearest-neighbor search (also called similarity search) is the problem of preprocessing a set S of n objects (which we henceforth refer to as *points*) lying in some space U with a similarity function so that given a query $q \in U$, one can efficiently locate the point that is most similar to q among the points in S. There are several surveys of nearest neighbor search (NNS); e.g., [9, 8, 18, 21]. This intensively studied problem has numerous web-related applications besides playing a central role in many other important areas. These applications include:

- Near-duplicate detection [4, 7, 15, 20].
- Classification/filtering tasks like detecting spam pages [37], spelling correction [11], sense disambiguation[36].
- Recommendation systems [34].
- Computing co-occurrence and co-citation similarity [6, 12].
- Discovering related tags in folksonomies [13, 32].
- Social network analysis (e.g. suggesting new friends) [32].
- News aggregation (searching for news articles that are most similar to the user's profile of interests) [26, 32].
- Advertisement targeting (searching for the most relevant website for displaying a given advertisement) [33].

Applying classical similarity search solutions for these tasks is not straightforward for several reasons. First, data models are quite *heterogeneous* and far from simple abstractions like Euclidean space. Combination of text data and link structure [3] is now a norm. For instance, consider a description of a blog: language, geographic location (dictionary parameters), age, number of posts (numerical parameters), referring links and reader list (graph parameters), text of profile and posts (text parameters) and posting timestamps (time parameters). Such a heterogeneous description leads to quite complicated similarity function. E.g. it can include manually defined logical rules and threshold functions. Finally, definition of similarity usually contains various "customization" parameters which can be adjusted by domain experts.

Second, many applications are focused on *similarity, not distance*. Mapping similarity values to distance values is tricky since we need to guarantee the triangle inequality (as most NNS-algorithms seem to crucially use the triangle inequality). Moreover, if most objects have low similarity with each other (as is often the case in high-dimensional spaces) almost all distances are in the range from 1/2 to 1. As a result, rules like "$|pp'| \geq 2|qp|$" (e.g. [8]) (which implies that p' is farther to q than p) will never apply. Note also that even if we have an algorithm for approximate nearest neighbor for distances, the approximation factor generally does not hold for the original similarity values.

Third, the number of description parameters is overwhelming and collections we face in practice are very far from "random [families] of random sets" [19]. We cannot expect to find efficient NNS-algorithms for general sets with arbitrary similarity functions. Hence, instead of addressing the problem in general we should attempt to find an additional properties that hold for real data and support efficient and provably correct NNS-algorithms.

Combinatorial approach.

Complicated data models and the absence of the triangle inequality for similarity values leave very few tools for the NNS-problem. To our knowledge, there are no NNS-algorithms that work directly with similarity values without first modelling the problem in a metric space. In the present paper, we propose a framework for designing algorithms that do not translate the problem to metric spaces.

Besides working with the similarity function directly, we introduce a restriction on the way the similarity function can be used. We call a search algorithm *combinatorial* if only direct comparisons between two pairwise similarity values are allowed. As we will see, even working with this restriction we can design good algorithms. The restriction offers some advantages: Since we make minimal assumptions about the similarity function, the algorithms in our framework have wide applicability. Moreover, the algorithms are quite robust to certain changes in the similarity function. E.g. adding a constant to all similarity values does not affect such an algorithm at all. Also, any transformation of the data representations that preserves the order of similarity values between the points, does not change anything in the output of the algorithm.

As mentioned before, we cannot expect efficient NNS-algorithms for general databases with arbitrary similarity functions. We introduce a natural property of databases which allows us to design efficient NNS-algorithms and is essentially satisfied for at least one real database, as described in our experimental results.

Consider a set S of n points. For every point x we sort all other point by their similarity to x. Overall, we have n sorted lists of $n-1$ points each. This is the only information actually to be used in the preprocessing stage. We denote the position of point x in y's list by $\mathrm{rank}_y(x)$.

Now we introduce notion of *nice* sets for nearest neighbor search. Nice sets should satisfy the following informal principle: For every triple of points x, y, z, if x has small rank with respect to z, and y has small rank with respect to z then x also has quite small rank with respect to y. This is our replacement for the triangle inequality. Actually, we relax it by a multiplicative factor D and call it *disorder inequality*:

$$\mathrm{rank}_y(x) \leq D \cdot (\mathrm{rank}_z(x) + \mathrm{rank}_z(y)).$$

We observe that *disorder constant* D is connected to the concept of "intrinsic dimension" for metric spaces. More precisely, $\log_2 D + 1$ can be used as a fair estimate of the real dimension of uniformly distributed sets in Euclidean space of some fixed dimension. Hence, we call $\log_2 D + 1$ the *disorder dimension* of the set.

It is natural to conjecture that if the disorder constant (and hence, the disorder dimension) is small, then the nearest neighbors problem has provably more efficient solutions than brute force. Thus our plan is (1) to verify the "small disorder assumption" on some real data set and (2) to construct a search algorithm utilizing this assumption.

Results.

Our first contribution is the introduction of the concepts of *combinatorial algorithm*, *disorder inequality* and *disorder dimension*. To the best of our knowledge this is the first time that a framework for comparison-based similarity search has been explicitly introduced.

Next, we present two new randomized algorithms for exact nearest neighbor search: Ranwalk and Arwalk. They are the first ones known to be *purely combinatorial* in the sense defined above. They bear some resemblance to the greedy search algorithms for small world networks (see, e.g., [24]).

Ranwalk performs a random walk in the search phase. It requires $\mathcal{O}(n^2)$ preprocessing space, $\mathcal{O}(n^2 \log n)$ preprocessing time and uses expected $\mathcal{O}(D \log n \log\log n + D^2)$ time for answering a query. It always produces the correct answer.

The Arwalk algorithm (walk via navigation array) requires $\mathcal{O}(nD \log n \log\log n)$ preprocessing space, $\mathcal{O}(n^2 \log n)$ preprocessing time and uses $\mathcal{O}(D \log n(\log\log n + log 1/\delta) + D^2)$ time for answering a query. For every query it produces the correct answer with probability at least $1 - \delta$. The underlying data structure, called *navigation array*, is a $n \times D' \times \log n$ table of pointers to points in the database S. Informally, for every point $x \in S$ and every $k \leq \log_2 n$ we keep pointers to $D' = D \log\log n$ random points in the $n/2^k$ neighborhood of x.

The analysis of Arwalk shows that similarity search is tractable (near-linear preprocessing space, near-logarithmic query time) when the disorder dimension $\log D + 1$ is at most $\log\log n$. Thus, we have similar results to [9, 10, 27, 28, 23] where the tractability was shown under the $\log\log n$ bound for other definitions of intrinsic dimension. One important advantage of Ranwalk and Arwalk is that they are the only *exact* algorithms in this family.

On the experimental side, we compute the disorder dimension of Reuters corpus [31]. The results show that (1)

on average the disorder inequality requires a fairly small constant D, (2) even for triangles with large rank values on the sides, the average D is still small, but (3) *in the worst case* for some triangles, the disorder fraction is large.

Outline.

The next section is devoted to our concept of disorder. Then we present our algorithms: Ranwalk and Arwalk. The experiments on the Reuters corpus are presented in Section 4. Finally, we explain the relation of our results to previous research and present six directions for further research.

2. COMBINATORIAL FRAMEWORK AND DISORDER INEQUALITY

As mentioned earlier, one of the main contribution of the present work is the combinatorial framework for the nearest neighbor problem. To describe this framework we need some notation.

DEFINITION 1. *(similarity space) A* similarity space *is a tuple (U, σ), where U is a finite universe of points, and U is equipped with a similarity function $\sigma : U \times U \to \mathbb{R}$.*

Intuitively, if $\sigma(p, q)$ is large for $p, q \in U$, then p and q are more similar to each other than when $\sigma(p, q)$ is small. Thus similarity has the opposite interpretation of distance in a metric space. Note that we do not require the similarity function to be symmetric.

The nearest neighbor problem is the following: given a database $S \subseteq U$, we would like to preprocess it so that for any query $q \in U$ we are able to find a point in S that is closest to q; that is to say, we would like to find $p \in S$ such that $\sigma(q, p) = \min\{\sigma(q, p) \mid p \in S\}$. Note that the similarity function need not be symmetric.

In our combinatorial framework, we propose to do away with the numerical value of the similarity function. Instead, we distill the problem down to the relative values of the similarity function. To make this precise we introduce the notion of *rank*.

One advantage of our framework is that any transformation of the set, or change in similarity function, that does not affect the rank function also does not affect the output of the nearest neighbor search. For example, if one adds a fixed number to all similarity values, an algorithm in our framework will work precisely in the same way. Note that classical ANN algorithms do not have this property.

DEFINITION 2. *(rank function) For points x, y and set S define $rank_{x,S}(y)$ to be the position of y when the elements z of $S \cup \{x, y\}$ are ordered according to decreasing similarity to x. When the set S is clear from the context we abbreviate $rank_{x,S}(y)$ to $rank_x(y)$.*

Let's restate the nearest-neighbor problem. In a similarity space (U, σ), given a database $S \subseteq U$, preprocess S so that given a query $q \in U$, we can quickly find the nearest neighbor of q in S, that is $p \in S$ such that $\text{rank}_{q,S}(p) = 1$.

In our combinatorial framework, the only operations an algorithm can do are to compute the similarity values and compare them with each other; in the preprocessing step, the algorithm can store the outcomes of these operations. We cannot expect to find efficient nearest-neighbor algorithms for general similarity spaces, and thus we need to find out some properties that are likely to be satisfied in real databases, and which also admit efficient nearest neighbor algorithms. We identify one such property next.

But before we do that, let's consider an example to motivate the definition. Suppose that set S is an (infinite) set of equally distant points on a line. Let us define the similarity function between two points to be the reciprocal of the Euclidean distance between them. Then it is easy to verify that for any triple $x, y, z \in S$ we have

$$\text{rank}_y(x) \le \text{rank}_z(x) + \text{rank}_z(y) + 3.$$

Similarly we have three more inequalities, if we replace $\text{rank}_z(x)$ by $\text{rank}_x(z)$ and/or $\text{rank}_z(y)$ by $\text{rank}_y(z)$ in the above inequality.

Of course, for a general set of points this will not be satisfied. Disorder constant $D(S)$ of a set of points is a measure of how far a given set of points S is from a total order:

DEFINITION 3. *(disorder constant D) Let S be a set equipped with corresponding similarity function σ. Define $D(S)$, the* disorder constant *of S, to be the smallest number D such that for all triples $x, y, z \in S$ we have the following* disorder inequality*:*

$$rank_y(x) \le D(rank_z(x) + rank_z(y)). \tag{1}$$

We often abbreviate $D(S)$ to D, when S is clear from the context. Other variants of this definition may also be useful; e.g., we may also require three "brothers" of the above inequality to be satisfied:

$$\text{rank}_y(x) \le D(\text{rank}_x(z) + \text{rank}_z(y)), \tag{2}$$
$$\text{rank}_y(x) \le D(\text{rank}_x(z) + \text{rank}_y(z)), \tag{3}$$
$$\text{rank}_y(x) \le D(\text{rank}_z(x) + \text{rank}_y(z)). \tag{4}$$

It follows from (1) of the above inequalities that for all x, y we have

$$\text{rank}_x(y) \le D\text{rank}_y(x). \tag{5}$$

There is a natural notion of similarity attached to a metric space: we can think of the similarity between two points as reciprocal or negative of the distance between them. In either case, we get the same similarity ordering. As it turns out, sometimes a notion of dimension defined for the metric space is closely related to the disorder constant for its associated similarity order. We now show that for grids $\log_2 D + 1$ is essentially the dimension of the grid.

LEMMA 1. *For the d-dimensional integer grid $\mathbb{Z}^d$ with Euclidean distance of sufficiently small step size, its disorder constant is 2^{d-1} upto a smultiplicative constant close to one.*

PROOF. Let us fix some step of Euclidean lattice and radius r_0. Let c be a constant such that the number of points in an *inner space* of ball of radius $r > r_0$ centered at a lattice point is lower upper bounded by cr^d. Let c' be a constant such that the number of points in a *ball with surface* of radius r is upper bounded by $c'r^d$. Actually, for any $\varepsilon > 0$ there is sufficiently small step of lattice and sufficiently large r_0 such that $c'/c \le 1 + \varepsilon$.

For points $x, y \in \mathbb{Z}^d$, denote by $|x, y|$, the Euclidean distance between x and y. For $x, y \in \mathbb{Z}^d$, we have

$$c(|x, y|)^d \le \text{rank}_x(y) \le c'(|x, y|)^d.$$

This gives

$$|z,x| \le \left(\frac{\mathrm{rank}_z(x)}{c}\right)^{1/d},$$
$$|z,y| \le \left(\frac{\mathrm{rank}_z(y)}{c}\right)^{1/d},$$
$$|y,x| \ge \left(\frac{\mathrm{rank}_y(x)}{c'}\right)^{1/d}.$$

Now, the triangle inequality $|y,x| \le |z,x| + |z,y|$, along with the above inequality yields

$$\left(\frac{\mathrm{rank}_y(x)}{c'}\right)^{1/d} \le \left(\frac{\mathrm{rank}_z(x)}{c}\right)^{1/d} + \left(\frac{\mathrm{rank}_z(y)}{c}\right)^{1/d}.$$

Raising both sides to dth power gives

$$\mathrm{rank}_y(x) \le \frac{c'}{c}\left(\mathrm{rank}_z(x)^{1/d} + \mathrm{rank}_z(y)^{1/d}\right)^d \le \frac{c'}{c}2^{d-1}(\mathrm{rank}_x(y) + \mathrm{rank}_y(z)).$$

Similarly, we can show that other inequalities in Def. 3 are satisfied. This shows that for $\mathbb{Z}^d$ we can take the disorder constant to be close to 2^{d-1} (because $\frac{c'}{c} \approx 1$). □

In Section 5 we mention further connections with metric spaces. The above lemma motivates the following definition.

DEFINITION 4. *(disorder dimension) Let (S,σ) be a set in similarity space and $D(S)$ be the disorder constant of S. Then we call* $1 + \log D$ disorder dimension *of S.*

3. NEAREST NEIGHBOR SEARCH IN SETS WITH SMALL DISORDER

In this section we present two related randomized algorithms for nearest neighbor search in similarity spaces (databases equipped with a similarity order) with disorder constant D. Both algorithms, called Ranwalk and Arwalk, use a walk through the database. Ranwalk always returns a correct answer but uses quadratic preprocessing space. Arwalk substantially reduces preprocessing space at the cost of allowing a small probability of error. Throughout the paper we assume that computing similarity value and comparing any two of them have a unit cost.

3.1 Ranwalk algorithm

Now we describe our random walk algorithm. The high level idea of the algorithm is to start at an arbitrary point $p \in S$, and then search its appropriate neighborhood for points that are more similar to the query point q, move to the best such point and repeat the process. The neighborhood in which we search can have a large number of points making it difficult to find points more similar to q. We resolve this difficulty by taking a small random sample of the neighborhood and choosing the point in the sample most similar to q.

We now formally present the algorithm, and then prove its correctness and analyze its performance. Let us set

$$D' := 3D(\log\log n + 1).$$

Ranwalk algorithm

Preprocessing:

- For every point x in database we sort all other points by their similarity to x. Thus our data structure consists of n lists of $n-1$ points each.

Query processing:

1. Step 0: choose a random point p_0 in the database.
2. From $k = 1$ to $k = \log n$ do Step k: Choose D' random points from $\min(n, \frac{3Dn}{2^k})$-neighborhood of p_{k-1}. Compute similarities of these points w.r.t. q and set p_k to be the most similar one.
3. If $\mathrm{rank}_{p_{\log n}}(q) > D$ go to step 0, otherwise search the whole D^2-neighborhood of $p_{\log n}$ and return the point most similar to q as the final answer.

THEOREM 1. *Assume that database points together with query point $S \cup \{q\}$ satisfy disorder inequality with constant D:*

$$rank_x(y) \le D(rank_z(x) + rank_z(y)).$$

Then Ranwalk algorithm always answers nearest neighbor queries correctly. It uses the following resources:
Preprocessing space: $\mathcal{O}(n^2)$.
Preprocessing time: $\mathcal{O}(n^2 \log n)$.
Expected query time: $\mathcal{O}(D \log n \log\log n + D^2)$.

PROOF. Indeed, claimed time and space constraints for preprocessing are satisfied.

We call a single execution of steps 0 to $\log n$ a *trip*. For every $k \le \log n$ we say that the first k steps *are successful* if $\mathrm{rank}_q(p_k) \le \frac{n}{2^k}$. Now we estimate the probability of success for the first k steps under the assumption that the first $k-1$ steps are, indeed, successful.

During the query processing the following lemma holds.

LEMMA 2. *Assume $rank_q(p_{k-1}) \le \frac{n}{2^{k-1}}$. Then with probability $1 - 1/2\log n$ the next inequality also holds: $rank_q(p_k) \le \frac{n}{2^k}$*

PROOF. There are exactly $\frac{n}{2^k}$ points satisfying inequality $\mathrm{rank}_q(x) \le \frac{n}{2^k}$. For any of such points x disorder inequality implies

$$\mathrm{rank}_{p_{k-1}}(x) \le D(\frac{n}{2^{k-1}} + \frac{n}{2^k}) = \frac{3Dn}{2^k}.$$

Thus, in $\frac{3Dn}{2^k}$ range from p_{k-1} the fraction of *good* points equals to $1/3D$. Since we use $D' = 3D(\log\log n + 1)$ random pointers of level k, there is at most $1/2\log n$ chance that we miss all good points. □

The above lemma states an upper bound for probability of not reducing the rank with respect to q by a factor of half in a single step. Hence, summing up error probabilities for all $\log n$ steps of a single trip, we have probability at least $1/2$ for trip success: $\mathrm{rank}_q(p_{\log n})$ is at most $\frac{n}{2^{\log n}} = 1$.

By disorder inequality this implies that $\text{rank}_{p_{\log n}}(q) \le D$. Therefore, a single trip has probability at least $1/2$ to achieve stopping condition. And hence, expected number of trips is at most 2.

Now let's consider the final "refinement procedure". Let $\text{rank}_p(q) \le D$ and let p' be true nearest neighbor for q, that is $\text{rank}_q(p') = 1$. Thus, $\text{rank}_{p'}(q) \le D$. Applying disorder inequality to triple p, p', q, we get $\text{rank}_p(p') \le D^2$. Thus exhaustive search of D^2 neighborhood guarantees the correct answer. This completes the analysis of Ranwalk. □

3.2 Arwalk algorithm

We now present our second algorithm called Arwalk (walk via navigation array). It achieves better preprocessing space complexity but sometimes makes mistakes (i.e. it is not zero-error algorithm). However, error probability δ can be arbitrarily decreased under reasonable increase of used resources.

The new algorithm is based on the first one, and is obtained by making the random choices of our basic algorithm in advance.

Let us set

$$D' = 3D(\log\log n + \log 1/\delta).$$

Our data structure consists of D' pointers for every of $\log n$ levels for every of n points. We give *navigation array* name to this $n \times \log n \times D'$ table.

Arwalk algorithm

Preprocessing:

- For every point x in database we sort all other points by their similarity to x. For every *level number* k from 1 to $\log n$ we store pointers to D' random points within $\min(n, \frac{3Dn}{2^k})$ most similar to x points.

Query processing:

1. Step 0: choose a random point p_0 in the database.
2. From $k = 1$ to $k = \log n$ do Step k: go by p_{k-1} pointers of level k. Compute similarities of these D' points to q and set p_k to be the most similar one.
3. Return $p_{\log n}$.

THEOREM 2. *Assume that database points together with query point $S \cup \{q\}$ satisfy disorder inequality with constant D:*

$$rank_x(y) \le D(rank_z(x) + rank_z(y)).$$

Then for any probability of error δ Arwalk algorithm answers nearest neighbor query within the following constraints:
Preprocessing space: $\mathcal{O}(nD\log n(\log\log n + \log 1/\delta))$.
Preprocessing time: $\mathcal{O}(n^2 \log n)$.
Query time: $\mathcal{O}(D\log n(\log\log n + \log 1/\delta))$.

PROOF. Computing pointers for every point require $\mathcal{O}(n\log n) + D'\log n$ time. Doing it sequentially we can manage to use only space proportional to output, t hat is $\mathcal{O}(nD'\log n)$. Thus, claimed time and space constraints for preprocessing are satisfied.

During the query processing the following lemma is satisfied.

LEMMA 3. *Assume $rank_q(p_{k-1}) \le \frac{n}{2^{k-1}}$. Then with probability $1-\delta/\log n$ the next inequality also holds: $rank_q(p_k) \le \frac{n}{2^k}$*

PROOF. There are exactly $\frac{n}{2^k}$ points satisfying inequality $\text{rank}_q(x) \le \frac{n}{2^k}$. For any of such points x disorder inequality implies

$$\text{rank}_{p_{k-1}}(x) \le D(\frac{n}{2^{k-1}} + \frac{n}{2^k}) = \frac{3Dn}{2^k}.$$

Thus, in $\frac{3Dn}{2^k}$ range from p_{k-1} the fraction of *good* points equals to $1/3D$. Since we use $D' = 3D(\log\log n + \log 1/\delta)$ random pointers of level k, there is at most $\delta/\log n$ chance that we miss all good points. □

Summing up error probabilities for all levels, we have at most δ probability that $rank_q(p_{\log n})$ is larger than $\frac{n}{2^{\log n}} = 1$. This completes the analysis of Arwalk. □

Remark. By additional cost of D^2 we can make Arwalk to be one-side error algorithm. That is, we can do the same "stopping condition" and "refinement procedure" as in Ranwalk. If stopping condition is satisfied we guarantee the correctness of answer. If it failed we return "Sorry, it was a bad luck. Here is the answer, but it does not seems to be a true nearest neighbor".

Discussion.

Let us summarize important properties of Ranwalk and Arwalk algorithms:

- Both are exact algorithm. That is, the objective of algorithm is to output the nearest neighbor, not the second or the third one.
- Ranwalk has deterministic preprocessing and randomized query processing. Arwalk has randomized preprocessing and deterministic query processing. The latter is a version of the random walk algorithm where all random choices are done in advance.
- For Arwalk, probability of error is taken *entirely* from random choices of navigation array. Unlike many works (e.g., [19]), we do not assume any particular distribution on query and/or database.
- Decreasing probability of Arwalk's error has logarithmic cost. E.g. to obtain probability error equal to $1/n$ we have to increase space and time by a factor of $\log n$.

4. EXPERIMENTAL EVALUATION

In this section, we evaluate the concept of disorder dimension and the ranwalk algorithm experimentally. For evaluation, we use the problem of nearest neighbor search in the Reuters-RCV1 corpus [31]. Text classification is an important application of nearest neighbor search and the RCV1 corpus is one of the most widely used data sets for evaluating text classification performance. RCV1 has roughly one gigabyte of text. It consists of about 800,000 documents that were sent over the Reuters newswire during a one year

period between August 20, 1996, and August 19, 1997. The collection covers a wide range of topics, including politics, business, and sports. The labels assigned to documents by Reuters editors can be used as a gold standard to evaluate nearest neighbor algorithms.

Disorder constant D.

In the previous section we introduced two algorithms, whose query time and preprocessing space depend upon the disorder constant D. These theoretical estimates become practical only if D is sufficiently small. Hence, a natural question arises: How small is D for real data sets?

The value of D as defined in Def. 3 is in a sense a worst-case definition: we want the inequality (1) to hold for all triples. This worst case definition leads to D being too large. However, as we will see presently, in the case of Reuters D it is small for most pairs.

We run a large number of *micro-experiments* with parameters $x \in S$ and $R \in \mathbb{Z}$ on the first 1000 documents of Reuters. Choose a, b in $[1, \ldots, R]$ uniformly at random. Consider a randomly chosen news article z. Let x and y be the articles with $\text{rank}_z(x) = a$ and $\text{rank}_z(y) = b$. Let $c := \text{rank}_y(x)$. Finally, compute $\frac{c}{a+b}$. This ratio corresponds to D for triple (x, y, z) using inequality (3).

Our experiments indicate that for $R = 2, 5, 10, 50, 100, 1000$, the ratio $\frac{c}{a+b}$ is no more than 200, for an overwhelmingly large fraction of the cases examined. And, in fact, for a large fraction of cases the ratio was close to 1. We illustrate this for $R = 5$ in Fig. 1.

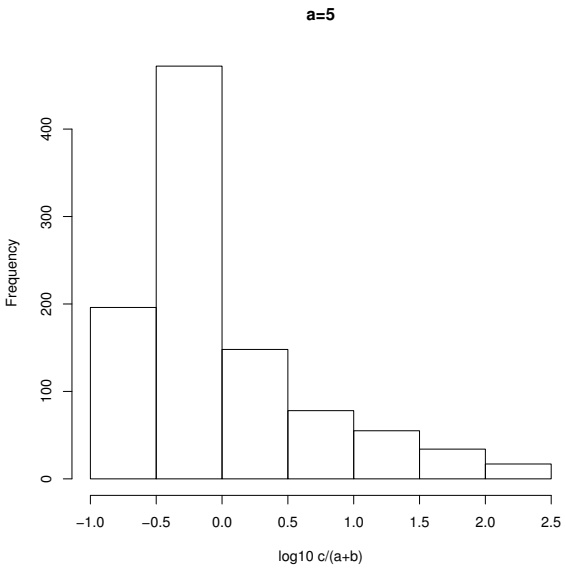

Figure 1: Distribution of $\text{rank}_y(x)/(\text{rank}_z(x) + \text{rank}_z(y))$ for $R = 5$

A second test of the disorder constant is to compute the ratio $\text{rank}_y(x)/\text{rank}_x(y)$ for pairs $< x, y >$, where x is randomly chosen from S and y is randomly chosen from the R closest neighbors of x. If values of this ratio are within a small interval, then this indicates that the disorder constant approximately reflects the true distribution of points. This does seem to be the case for Reuters as shown in Fig. 2 for $R = 5$ (the graph shows a histogram for 10,000 trials). Almost all ratios r are $10^{-1} \leq r \leq 10^{0.3}$ and none are $< 10^{-1.2}$. We obtained similar results for $R = 2, 5, 10, 50, 100, 1000$.

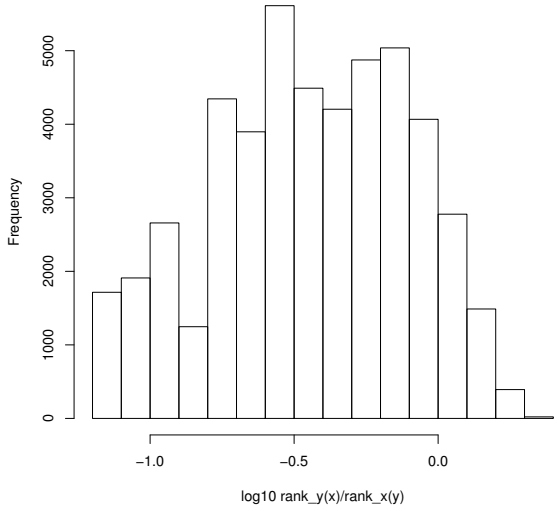

Figure 2: Distribution of $\text{rank}_y(x)/\text{rank}_x(y)$ for $R = 5$

Ranwalk algorithm.

We also evaluated the performance of the ranwalk algorithm on Reuters. We chose the parameter $D = 10$ and selected a set of 1000 documents as training set and a disjoint set of 1000 documents as test set. We then performed 10,000 trials of randomly selecting a test point q and executing ranwalk to find the nearest neighbor p. For some points q, $\text{rank}_p(q) > D$ for all training points p since the disorder relationship does not hold for all points in the database. We therefore limited the number of "restarts" (i.e., returning to step 0 after failing to find a p with $\text{rank}_p(q) \leq D$) to 100. In those cases, we selected from the set of 100 $p_{\log n}$ candidates that p in step 3 that was closest to q.

Fig. 3 shows the results of the experiment. For almost all query points, the algorithm successfully identified the nearest neighbor (note logarithmic scale). There was only 1 case with $\text{rank}_q(p) \geq 30$. These results indicate good performance of Ranwalk on practical nearest neighbor search problems.

5. RELATED WORK AND OPEN PROBLEMS

Related work.

Our work is inspired by several ideas presented in previous publications. The first one is to use walks and, in particular *random walks* for search problems. For nearest neighbors a walk algorithm was suggest by Orchard [38] – however this algorithm is not randomized. Also, this idea was used in SAT algorithms [17], for measuring index quality [16] and for ranking nodes in the Web graph [1].

The second idea is the *small disorder constant assumption.*

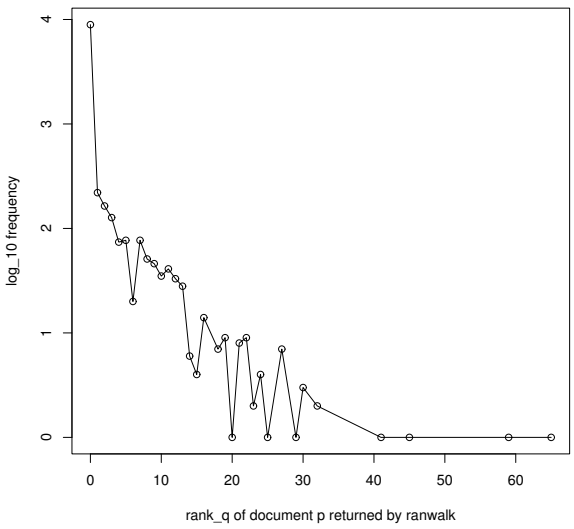

Figure 3: Distribution of $\text{rank}_q(p)$ of p returned by ranwalk

The informal idea that small dimensional spaces (for some appropriate notion of dimension) are more nicely behaved and tractable, is now well-known in many settings. In particular, for nearest neighbor search it has appeared in several recent papers, where various notions of the dimension of a metric space are introduced, and efficient algorithms are given for small dimensional spaces: e.g., KR-dimension, doubling dimension [9, 10, 23, 27, 28].

DEFINITION 5. *KR-dimension of a metric space M is the minimum d such that $|B_{2r}(x)| \leq 2^d |B_r(x)|$, for all x and r; here $B_r(x)$ denotes the ball of radius r with center x.*

We are currently unable to prove any direct comparison result for disorder dimension vs KR-dimension or doubling dimension. However, one modelling example shows an advantage of disorder dimension over KR one. Consider a set of points where almost all distances (dissimilarity values) are in the interval $[1/2, 1]$. Then take any point x, let y be it's nearest neighbor and say $d(x, y) = 0.6$. Then by taking radius 0.6 we have just one point in the ball but by taking 1.2 we get the whole database. Hence, KR-dimension is the maximal possible for n points in this example. On the other hand, there is still a chance for disorder to be small.

Finally, the principle "respective order but not absolute values are really important" was previously applied in the somewhat related setting of Web ranking problem [30]. They were concerned with how different Web ranking algorithms change the ranks of the webpages because of small changes in the input.

Next steps.

We feel that the idea of focusing on rank values instead of similarity values is quite promising. Hence, we suggest to continue the study of nearest neighbors in this direction. Here is our question list:

Average disorder. If disorder inequality does not hold for a small fraction of pairs, how should we modify our algorithm? One possible approach is to use some other method (e.g. inverted index [35]) for determining a "relatively similar" point in the database and then start the random walk from this point.

Improving our algorithms. How can one decrease preprocessing complexity of our algorithms? Is it possible to combine advantages of Ranwalk and Arwalk? Does there exist a deterministic algorithm with sublinear search time utilizing small disorder assumption? E.g., can we use expanders for derandomization? Can we use Ranwalk/Arwalk under other than disorder-based assumptions?

Disorder of random sets. Compute disorder values for some modelling examples. For example, consider n random points on d-dimensional sphere, or n random strings of some fixed length in σ-size alphabet for Hamming/edit distance.

Further experiments. Compute disorder constant for other data sets. Implement and test Ranwalk and Arwalk. Study them in the context of MESSIF project [5].

Lower bounds. Is it possible to prove lower bounds on preprocessing and query complexities in some "black-box" model of computation? Can we adapt techniques from recent papers [28, 39]?

Utilizing combinatorial framework. Consider well-known techniques like search trees [8, 18], hashing [2, 22, 29] or random projections [14, 25] under bounded disorder assumption. What are their "combinatorial" analogues? Can they beat random walk? Construct disorder-inequality-based clustering algorithms. Also, what is analogue of disorder inequality for bichromatic nearest neighbor search?

6. REFERENCES

[1] A. Agarwal and S. Chakrabarti. Learning random walks to rank nodes in graphs. In *ICML'07*.

[2] A. Andoni and P. Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. In *FOCS'06*, 2006.

[3] R. Angelova and G. Weikum. Graph-based text classification: Learn from your neighbors. In *SIGIR'06*.

[4] Z. Bar-Yossef, I. Keidar, and U. Schonfeld. Do not crawl in the dust: Different urls with similar text. In *WWW'06*.

[5] M. Batko, D. Novak, and P. Zezula. MESSIF: Metric similarity search implementation framework. In *DELOS'07*.

[6] A. Blessing and H. Schütze. Inverted indexes for fast distributional similarity computations. To appear, 2007.

[7] A. Z. Broder, M. Charikar, A. M. Frieze, and M. Mitzenmacher. Min-wise independent permutations. In *STOC'98*.

[8] E. Chávez, G. Navarro, R. Baeza-Yates, and J. L. Marroquín. Searching in metric spaces. *ACM Computing Surveys*, 2001.

[9] K. L. Clarkson. Nearest-neighbor searching and metric space dimensions. In Nearest-Neighbor Methods for Learning and Vision: Theory and Practice, MIT Press, 2006.

[10] R. Cole and L.-A. Gottlieb. Searching dynamic point sets in spaces with bounded doubling dimension. In *STOC'06*.

[11] R. Cole, L.-A. Gottlieb, and M. Lewenstein. Dictionary matching and indexing with errors and don't cares. In *STOC '04*, pages 91–100, 2004.

[12] J. Dean and M. R. Henzinger. Finding related web pages in the world wide web. In *WWW'99*.

[13] M. Dubinko, R. Kumar, J. Magnani, J. Novak, P. Raghavan, and A. Tomkins. Visualizing tags over time. In *WWW'06*.

[14] R. Fagin, R. Kumar, and D. Sivakumar. Efficient similarity search and classification via rank aggregation. In *SIGMOD'03*.

[15] D. Fetterly, M. Manasse, and M. Najork. Detecting phrase-level duplication on the world wide web. In *SIGIR'05*.

[16] M. Henzinger, A. Heydon, M. Mitzenmacher, and M. Najork. Measuring index quality using random walks on the web. In *WWW'99*.

[17] E. A. Hirsch and A. Kojevnikov. Unitwalk: A new SAT solver that uses local search guided by unit clause elimination. *Annals of Mathematics and Artificial Intelligence*, 2005.

[18] G. Hjaltason and H. Samet. Index-driven similarity search in metric spaces. *ACM Transactions on Database Systems*, 2003.

[19] B. Hoffmann, Y. Lifshits, and D. Nowotka. Maximal intersection queries in randomized graph models. In *CSR'07*.

[20] S. Ilyinsky, M. Kuzmin, A. Melkov, and I. Segalovich. An efficient method to detect duplicates of web documents with the use of inverted index. In *WWW'02*.

[21] P. Indyk. Nearest neighbors in high-dimensional spaces. Chapter 39 of Handbook of Discrete and Computational Geometry, CRC Press, Second Edition, 2004.

[22] P. Indyk and R. Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In *STOC'98*.

[23] D. R. Karger and M. Ruhl. Finding nearest neighbors in growth-restricted metrics. In *STOC'02*.

[24] J. Kleinberg. The small-world phenomenon: an algorithm perspective. In *STOC '00: Proceedings of the thirty-second annual ACM symposium on Theory of computing*, pages 163–170, New York, NY, USA, 2000. ACM Press.

[25] J. M. Kleinberg. Two algorithms for nearest-neighbor search in high dimensions. In *STOC'97*, pages 599–608, 1997.

[26] J. A. Konstan, B. N. Miller, D. Maltz, J. L. Herlocker, L. R. Gordon, and J. Riedl. Grouplens: applying collaborative filtering to usenet news. *Commun. ACM*, 40(3):77–87, 1997.

[27] R. Krauthgamer and J. R. Lee. Navigating nets: simple algorithms for proximity search. In *SODA'04*.

[28] R. Krauthgamer and J. R. Lee. The black-box complexity of nearest-neighbor search. *Theoretical Computer Science*, 2005.

[29] E. Kushilevitz, R. Ostrovsky, and Y. Rabani. Efficient search for approximate nearest neighbor in high dimensional spaces. *SIAM J. Comput.*, 2000.

[30] R. Lempel and S. Moran. Rank-stability and rank-similarity of link-basedweb ranking algorithms in authority-connected graphs. *Information Retrieval*, 2005.

[31] D. D. Lewis, Y. Yang, T. G. Rose, and F. Li. RCV1: A new benchmark collection for text categorization research. *Journal of Machine Learning Research*, 2004.

[32] Y. Lifshits. A guide to web research. Materials of mini-course at Stuttgart University. Available at `http://logic.pdmi.ras.ru/~yura/webguide.html`, 2007.

[33] Y. Lifshits and D. Nowotka. Estimation of the click volume by large scale regression analysis. In *CSR'07*.

[34] G. Linden, B. Smith, and J. York. Amazon.com recommendations: item-to-item collaborative filtering. *Internet Computing*, 2003.

[35] C. D. Manning, P. Raghavan, and H. Schütze. *Introduction to Information Retrieval*. Cambridge University Press, to appear in 2008.

[36] C. D. Manning and H. Schütze. *Foundations of Statistical Natural Language Processing*. MIT Press, 1999.

[37] A. Ntoulas, M. Najork, M. Manasse, and D. Fetterly. Detecting spam web pages through content analysis. In *WWW'06*.

[38] M. T. Orchard. A fast nearest-neighbor search algorithm. In *ICASSP'91*.

[39] C. Sahinalp and A. Utis. Hardness of string similarity search and other indexing problems. In *ICALP'04*.