Finding Near Neighbors Through Cluster Pruning

Flavio Chierichetti [*]
Computer Science Department, University of Rome "La Sapienza"
chierichetti@di.uniroma1.it

Alessandro Panconesi [*]
Computer Science Department, University of Rome "La Sapienza"
ale@di.uniroma1.it

Prabhakar Raghavan
Yahoo! Research
pragh@yahoo-inc.com

Mauro Sozio [*]
Computer Science Department, University of Rome "La Sapienza"
sozio@di.uniroma1.it

Alessandro Tiberi [*]
Computer Science Department, University of Rome "La Sapienza"
tiberi@di.uniroma1.it

Eli Upfal
Computer Science Department, Brown University
eli@cs.brown.edu

ABSTRACT

Finding near(est) neighbors is a classic, difficult problem in data management and retrieval, with applications in text and image search, in finding similar objects and matching patterns. Here we study *cluster pruning*, an extremely simple randomized technique. During preprocessing we randomly choose a subset of data points to be *leaders*; the remaining data points are partitioned by which leader is the closest. For query processing, we find the leader(s) closest to the query point. We then seek the nearest neighbors for the query point among only the points in the clusters of the closest leader(s). Recursion may be used in both preprocessing and in search. Such schemes seek approximate nearest neighbors that are "almost as good" as the nearest neighbors. How good are these approximations and how much do they save in computation?

Our contributions are: (1) we quantify metrics that allow us to study the tradeoff between processing and the quality of the approximate nearest neighbors; (2) we give rigorous theoretical analysis of our schemes, under natural generative processes (generalizing Gaussian mixtures) for the data points; (3) experiments on both synthetic data from such generative processes, as well as on from a document corpus, confirming that we save orders of magnitude in query processing cost at modest compromises in the quality of retrieved points. In particular, we show that p-spheres, a state-of-the-art solution, is outperformed by our simple scheme whether the data points are stored in main or in external memory.

Categories and Subject Descriptors

H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval—*clustering, search process*; F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems—*sorting and searching, geometrical problems and computations*

General Terms

Algorithms, Experimentation, Performance

Keywords

Clustering, Nearest Neighbor, Generative Model

1. INTRODUCTION

We are given a set $S = \{p_1, \ldots, p_n\}$ of n points in $d-$dimensional space. Given a new *query* point q, we seek to find the K points in S nearest to q, under some specified metric. This K*-nearest neighbors problem* arises persistently in text and multimedial retrieval, pattern matching, supervised learning/classification and data mining. Efficient solutions to this problem are both critical and few.

In these applications the dimensions are *features* arising from documents, images, media, fingerprints or other objects. Each object is turned into a point in the space of features, as is a query. Retrieval then reduces to finding the nearest neighbors to the query point. Because the number of features is almost always large, the dimensionality of the space is too large to invoke methods from computational geometry such as Voronoi partitions [14]. Consequently, it is difficult in the worst case to do much better than the following naive algorithm: measure the distance from the query point to each object point, then select the K shortest distances. In this scheme, the exhaustive computation of distances from the query to every data point is done at query time, rendering it a crucial bottleneck in query processing; this is often referred to as the *curse of dimensionality.*

To avoid such exhaustive distance computation and to speed up query processing, the key is to quickly eliminate from consideration many of the data objects; such *pruning* is the goal of any indexing scheme. Here we study an extremely simple pruning scheme known as *cluster pruning*, defined below in Section 1.1. The scheme is randomized, and may sometimes produce *near neighbors* that are not the nearest neighbors. How good are these near neighbors and how much computation does the scheme save? We present

[*]Research partially supported by a grant of Yahoo! Research and by the EC project DELIS.

both theoretical analysis and experiments showing that cluster pruning saves significant computation at modest compromises in the quality of the retrieved objects.

1.1 The technique

We first describe the basic scheme, referred to as *cluster pruning*, and then describe extensions of two forms: (1) recursion and (2) fanout.

The scheme entails two phases: *preprocessing* of the data set and *query processing*. During preprocessing we choose $\sqrt{n}$ data points at random; we call the chosen points *representatives* and denote them by $r_1, \ldots, r_{\sqrt{n}}$. Every other data point is then "attached" to the nearest representative; this induces a partition of the data points into $\sqrt{n}$ clusters, with one representative for each cluster. (One could view this as an extremely simple clustering scheme.) Finally, a *leader* is selected for every cluster. Let $\ell_1, \ldots, \ell_{\sqrt{n}}$ denote the set of leaders. We try three different ways to select leaders: (a) the leaders are the representatives; (b) they are the centroids of the corresponding clusters, or (c) the medioids of the clusters (i.e. the data point in each cluster closest to its centroid). We will see that the centroid option turns out to be the best and works remarkably well.

During query processing, we find the leader ℓ that is closest to the query by computing the distance from the query q to each of the $\ell_1, \ldots, \ell_{\sqrt{n}}$. We then seek the nearest neighbors for the query point from among only the points in the cluster of which ℓ is the leader. Thus, we avoid computing the distance from q to all other points, effectively *pruning* them away. Intuitively the nearest neighbors to the query point lie mainly in the cluster of the closest leader.

We demonstrate theoretically as well as experimentally the surprising effectiveness of this scheme. The simplicity of the scheme is one of its main virtues because it makes it amenable to a rigorous mathematical analysis. We show how the dimensionality of the space affects the performance of the method. Roughly, while the method is always efficient when the number of dimensions is low, it may generate very large clusters if the number of dimensions is order of $\log n$ or more. However, the method appears to be very reliable with real data even when the number of dimensions is very high (of the same order of n). We introduce a novel generative model for text data to explain this behavior. The model is dubbed the *hierarchical mixture of Gaussians*. We prove rigorously as well as experimentally that when data are generated by this model the method is both efficient and accurate in retrieval. These results are complemented by our experiments with real text corpora, for which our simple method exhibits remarkable performance. Not only does it behave very well when compared to the naive (but optimal) linear scan, it outperforms comparable solutions such as p-spheres and rank aggregation. We postpone a thorough discussion of our results to § 1.4.

We explore extensions of our basic scheme that provide interesting tradeoffs between search and I/O cost when data is stored in external memory. The first extension consists of using $L > 1$ levels of recursion in the scheme. Thus for $L = 2$, we would have the $n^{1/3}$ leaders at the top level, each of which in turn has $n^{1/3}$ sub-leaders, each representing a cluster. More generally, we recursively use a "fanout" of $n^{1/L+1}$ at each level, leading to the expected number of distance computations for a query being $(L+1)n^{1/L+1}$, at increased costs in random I/O requests to external memory.

Two other positive integer parameters a and b define the final family of extensions to the basic scheme. Following clustering in the preprocessing phase, we associate each data point to its nearest a representatives (rather than just one, as above). Secondly, during query processing we find the nearest b (rather than only one) leaders and seek the nearest neighbors to q from their clusters. While the parameter a increases the number of data points in a leaf, the parameter b leads to a more targeted search at an increased cost in I/O operations. We now present the pre-processing algorithms in more precise detail; the reader satisfied with the intuitive explanations above may skip Section 1.2 below.

1.2 Clustering Algorithms

Each algorithm has two parameters: L which is the height of the recursion, and a, the number of representatives each point is attached to.

Algorithm 1 Random Clustering

```
Input: A set $S$ of $n$ points, $L$ and $a$
Output: A clustering $C_1, C_2, \ldots, C_{n^{1/(L+1)}}$ of $S$
 1: for $i = 1$ to $L$ do
 2:   $\mathcal{L} \leftarrow \emptyset$
 3:   for $j = 1$ to $n^{1-i/(L+1)}$ do
 4:     $l_j \leftarrow$ a randomly chosen point in $S \setminus \mathcal{L}$
 5:     $\mathcal{L} = \mathcal{L} \cup \{l_j\}$, $C_j = \emptyset$ (cluster of $l_j$)
 6:   end for
 7:   for all $d \in S$ do
 8:     Let $C_1(d), \ldots, C_a(d)$ be the $a$ clst's closest to $d$
 9:     for $k = 1$ to $a$ do
10:       $C_k(d) \leftarrow C_k(d) \cup \{d\}$
11:     end for
12:   end for
13:   $S \leftarrow \emptyset$
14:   for $k = 1$ to $n^{1-i/(L+1)}$ do
15:     $S \leftarrow S \cup centroid(C_k)$
16:   end for
17: end for
18: return $C_1, \ldots, C_{n^{1/(L+1)}}$
```

Random Clustering. Given n points and positive integers L and a, this algorithm chooses uniformly at random $n^{1/(L+1)}$ representatives among the input points. Then, each point is attached to its a closest representatives. For each cluster thus built, its leader is computed, providing clusters each of $an^{1-1/(L+1)}$ points on average, to which the algorithm is applied recursively $L-1$ times. The reader may find it useful to picture the algorithm as building a tree of height L, with each node having $n^{1/(L+1)}$ children. The leaves of this tree are the "real" clusters, containing the input points.

Agglomerative Clustering. If random clustering performs well, might not some more sophisticated clustering algorithm perform even better? For this and other reasons we also studied agglomerative clustering in our context. Our agglomerative clustering algorithm is a computationally efficient variant of the one usually found in the literature, since the running time of the original makes it too costly to run on our text data set.

Processing a query. The computational effort spent processing a query depends on the parameter b: we match the query point against all leaders and then search inside the clusters of the closest b leaders. In terms of tree structure,

b governs how many leaves are exhaustively searched during query processing.

1.3 Framework and Metrics

Our study of this family of schemes must include measures for (1) the cost of query processing and (2) the quality of the approximate nearest neighbors found. This allows us to explore questions such as: for a given query processing cost (driven for instance by the query latency a user will tolerate), what is the best setting of parameters (such as L, a, b) to maximize quality?

To measure the query processing cost, we use the number of distance computations. In most applications the points $p_1, \ldots, p_n$ are unit length vectors; thus, finding the distance between two points is equivalent to computing the cosine similarity of the corresponding vectors. In these cases we use the number of cosine similarity computations as our measure of query processing cost. We also use a more refined metric to assess the cost of cosine similarity computations – the number of multiplications. The advantage of these measures is that they are relatively independent of implementation hardware. For memory-resident applications they are an accurate, machine independent, measure of running time. For disk-resident applications we will instead use the time spent performing I/O operations, expressed in milliseconds.

To measure the quality of the approximate nearest neighbors found by our schemes, we need the following definitions. Let $R_q := [x_1, x_2, \ldots, x_n]$ be the ordering (ranking) of the points in S by proximity to a query q. Let $G_q^K = \{x_1, \ldots, x_K\}$ be the *ground truth*, i.e., the prefix of R_q giving the K nearest neighbors to q. Let $A_q^K = \{a_1, \ldots, a_K\} \subset S$ be the set of K data points returned by algorithm A for the query q. Finally let $W_q^K = \{x_{n-K+1}, \ldots, x_n\}$ be the worst K documents according to the ordering R_q. Our first quality measure is *competitive recall at K:*

$$CR_K(\mathcal{A}) = \frac{|A_q^K \bigcap G_q^K|}{K}. \tag{1}$$

This simply asks what fraction of the ground truth is in A_q^K. Note that this is related to, but not quite the same, as the classic notion of recall in information retrieval (where recall is defined through a binary relevance judgment for each document, not by the selection of the top K).

We seek a second, more nuanced measure for the quality of points returned by a nearest-neighbor algorithm. Our philosophy here: in retrieval applications, cosine similarity between the query and a data point is a proxy for an end-user's perception of quality. Thus a point with cosine similarity 0.95 to the query is likely to be perceived as more useful than one with cosine similarity 0.45; but it is unlikely that a point of cosine similarity 0.95 is dramatically better than one of similarity 0.92, even though the former may be the nearest neighbor and the latter not.

Given a set B of documents (points, represented as unit-length vectors) let

$$s(B, q) := \frac{1}{|B|} \sum_{d \in B} q \cdot d$$

where the summation is over the dot products; for unit length vectors these are the same as cosine similarities. This is the average cosine similarity of a query q with respect to a set of points B. Let $\mathit{best}(q) := s(G_q^K, q)$ and $\mathit{worst}(q) := s(W_q^K, q)$. At first blush we may wish to use the ratio

$$r_q^K := \frac{s(A_q^K, q)}{\mathit{best}(q)}. \tag{2}$$

Instead we use the *(normalized) competitive similarity at K* for an algorithm[1] A:

$$CS_K(A) = \frac{s(A_q^K, q) - \mathit{worst}(q)}{\mathit{best}(q) - \mathit{worst}(q)}. \tag{3}$$

Being normalized with respect to the worst possible result, this metric is more informative and uniform than the ratio r_q^K. For instance it is possible to have $r_q^K = .99$ while $s(A_q^K, q) = \mathit{worst}(q)$, so a high value of r_q^K is not necessarily an indication of good performance. Note also that $CS_K(q) \leq r_q^K$; thus, an algorithm that performs well with respect to CS_K will give good results with respect to r_q^K as well; the two measures become identical as $\mathit{worst}(q) \to 0$.

1.4 Results

Analysis: A major contribution here is a mathematically rigorous analysis of random cluster pruning, presented in Section 2. A first question concerns efficiency. For $L = 1$ we expect clusters to be roughly balanced and have size $O(\sqrt{n})$. Thus we expect to perform only $O(\sqrt{n})$ distance computations when the search is restricted to $O(1)$ clusters, instead of the n required by a naive linear scan. (If we had Z leaders, we expect the query processing cost to be $O(Z + n/Z)$, which is minimized at $Z = \sqrt{n}$.) Perhaps surprisingly, we show that this intuition is sound for constant number of dimensions d, but it fails for $d = \Omega(\log n)$. We show that when d is constant then, with high probability, the method generates balanced clusters, i.e., $\sqrt{n}$ clusters, each having at most $\sqrt{n} \log n$ points. This result is independent of the distribution of the points. Second, we show that if $d = \Omega(\log n)$, there are data sets such that the method is likely to produce very large clusters. This also holds if we recursively apply random clustering for as long as large clusters persist, up to $O(\log n)$ iterations – a very counterintuitive behavior.

As a side remark, these results might hint at a fundamental difference between the geometry of low-dimensional corpora, such as image data, and that of high-dimensional corpora typical of text. In text retrieval applications the number of dimensions can be of the order of the number of points, and larger than its logarithm. Our results show that a clustering method that works well in low-dimension might behave very differently in high dimension. This might have some bearing on the experimental outcome of p-spheres and rank-aggregation, to be discussed below.

However, our experiments with real text corpora, to be described shortly, show that random clustering very reliably generates balanced clusters. This raises the question of what properties a corpus should have so that the clusters are balanced. To this end we introduce a stochastic generative model, the *hierarchical mixture of Gaussians* (HMG), and rigorously analyze our scheme for data generated by it. Our model is a plausible generative model for text-corpora since it captures the intuition that documents in a corpus belong to topics. Documents belonging to the same topic are more similar than documents belonging to different topics. Topics

[1] At slight overloading of notation we represent by A the set output by A.

are partitioned to sub-topics, with smaller variance, giving a more refined segmentation of the corpus.

We show that when the data points and queries are drawn from a HMG then, with high probability, (a) random clustering generates balanced clusters and (b) both competitive recall and competitive similarity tend to one as n becomes large, i.e., the method is both fast and also accurate. For concreteness we present the generative model in terms of documents falling into topics and sub-topics; however the model is general enough to represent other data such as images represented as vectors of features. Note that the algorithms are oblivious to the particular distribution; only the proofs depend on it. The technical core of these proofs relies on our ability to prove different bounds on the distances between points within a topic, versus points in separate topics. Perhaps the most interesting aspect of the analysis is that it works for such a simple scheme (since the algorithm is in no way "tweaked" to make the analysis easier). Both the model and the proof technique should be of independent interest in other machine learning applications especially for text. To this end we begin our analysis by establishing several mathematical propositions on the model, that should be useful for further research.

Experiments: We provide results on three classes of experiments. The first is on synthetic data drawn from the the HMG model. For these we achieve near-perfect retrieval quality, at a small fraction of exhaustive search cost. We then perform detailed experiments (Section 3) on a 95,000-document text data set – this appears to be large in both data size and dimensionality compared to many reported experiments. We consider both main and external memory settings. The main conclusion to be drawn from these experiments: with just 10% of the computational cost of exhaustive search, we retrieve better than 80% competitive recall and 90% (normalized) competitive similarity.

For in-memory applications the basic scheme, with no recursion (i.e., $L = a = 1$), gives best results. This is interesting because the basic scheme is space optimal (just the space needed to store $O(\sqrt{n})$ pointers for a data set of n documents). By increasing b, the number of clusters in which to continue our search, we can increase precision. For disk-resident applications, we pay b seeks, so it is crucial to keep this value low. In this case a larger value of a helps. Our experiments suggest that $a = 5$ gives the best results for external memory. Note the modularity of our scheme: a is precisely the space blowup, while b gives the number of seeks for disk I/O settings and it is also proportional to time in in-memory applications.

As mentioned, there are several ways to select leaders. In all our experiments centroids worked much better than medioids or representatives.

In the text experiments we tried long as well as short queries. Long queries consist of entire paragraphs, while short queries are made of just a few words. Not surprisingly the best results are obtained with long queries. We retain good quality with short queries however. Finally, and crucially, our pruning methods work very well already for small values of K.

We compare our simple scheme against p-spheres [25], reported to give good results for both in-memory and disk I/O applications. Among all state-of-the-art solutions for approximate nearest-neighbour computation, the comparison against p-spheres is especially meaningful because, roughly speaking, they are a random clustering targeted to a specific query distribution known in advance. In other words, by using p-spheres in the pre-processing phase one is trying to build clusters that are good with respect to the query distribution that is known a priori.

These tests are also interesting because they are, to the best of our knowledge, the first experimental study of p-spheres for data sets that typically arise in text applications: a large number of dimensions and very sparse vectors. The number of dimensions in the experiments reported in [25] was limited to a few dozens while in our experiments this number is several hundred thousands. The vectors in [25] were typically dense while here only a very tiny fraction of the coordinates is non zero.

Perhaps surprisingly, our simple scheme outperforms p-spheres. For the same computational effort, our scheme attains better quality in the case of in-memory applications and essentially the same quality for disk I/O. The main difference is space blow-up, which is very severe in the case of p-spheres (up to 17 times to achieve comparable quality and as large as 100 in some cases!). For in-memory applications the space blow-up of our scheme is negligible, while it is limited to 5 in the case of disk I/O. While space blow-up equals a for our scheme, in the case of p-spheres it is an unpredictable function of the various parameters of the pre-processing phase. Last but not least, to set up p-spheres one needs to know the query distribution in advance, while no such knowledge is needed for our scheme. Being independent of such knowledge is clearly desirable. Furthermore, as our experiments show, deviating from it yields a marked deterioration of the performance of p-spheres, while our scheme is unaffected.

We have also compared our scheme against another state-of-the-art solution: rank aggregation. Our method turned out to be far more accurate with our data set. We have not included the data here for lack of space.

1.5 Related prior work

The nearest neighbors problem has attracted significant interest in the database and algorithms communities in recent years [1, 2, 3, 4, 5, 6, 7, 8, 21, 22] . The difficulty in approaching the problem in high dimensional space has been demonstrated in [9] where it was shown that under broad classes of distributions the ratio of the largest and smallest distances in high dimensional space converges to 1 in probability. The argument works for fixed n as $d \to \infty$; it lacks a tail bound (or rate of convergence) as provided in our work, that is critical to any algorithm analysis. The insight of [9] is built on by [10], where a Bayesian decision rule is used to cluster data during pre-processing and the search phase scans a number of clusters for the nearest neighbors. Their analysis is restricted to a simple HMG (which, we will show in Section 2.2, corresponds to one level of our model) and assumes unspecified large dimension with respect to the number of data points.

Elaborate data structures are studied in Brin [11] where a complex tree structure is built during preprocessing. Search is done recursively in the data structure. No bounds on search time are given. In [12] a preprocessing phase builds m lists of n documents, each list containing a rank value for each document. In the query processing phase the algorithm extracts from these lists the approximate K-nearest neighbors. With $m = \Theta(\epsilon^2 \log n)$ the algorithm finds an ϵ-approximate $k-$nearest neighbor with high probability (a weaker quality measure than competitive recall and similarity). The expected query processing time is $O(n^{1-2/(m+2)})$; the experiments use two datasets of $10,000$ and $150,000$ objects but the number of dimensions is smaller than 800 in both collections.

Hashing is used in [13] to find approximate K-nearest neighbors. They give a theoretical analysis showing that the approximate K-nearest neighbors can be found with probability at least $1-\delta$ if the proposed algorithm is executed $O(1/\delta)$ times where δ is a constant greater than zero. The experimental results are limited to spaces with 64 dimensions.

Results on the vector space model and with cosine similarity to measure the similarity between documents can be found in [15], [16] and [17]. In [15] the authors develop a new clustering algorithm which is compared with k-means. Experimental results are given but no theoretical analysis.

In [16] an algorithm is given to improve running time in document search while keeping good recall guarantees. In a preprocessing phase a list of documents for each term is built. In the query processing phase query-document similarities are only partially calculated. The algorithm has been tested on a collection of roughly $13,000$ document abstracts but no theoretical analysis is given. Finally in [17] several clustering algorithms including k-means are compared; an additional variant called bisecting k-means is also considered. For a survey of clustering algorithms to improve document search see [18].

Other approaches to find the nearest neighbor use a generalized version of the B-tree [19] known as the R-tree [20] and its many variants (see, among others, the survey by Manolopoulos et al. [23].) These papers report experimental results but no analysis. In particular in [22] the authors propose a new index structure called the SR-tree to enhance performance on nearest neighbor queries. SR-trees are compared against p-spheres in [25] where the latter are reported to behave better.

2. THEORETICAL ANALYSIS

In this section we exploit the simplicity of random clustering, rigorously establishing some of its crucial properties. First, we show that if d, the number of dimensions is constant, then with high probability, the method generates balanced clusters, i.e. $\sqrt{n}$ clusters each with at most $\sqrt{n}\log n$ points each. Second, we show that if $d \geq \log n$, there are corpora such that the method is likely to produce very large clusters. This also holds if we recursively apply random clustering $O(\log n)$ times, a very counterintuitive behavior. However, the experiments in Section 3 show that random clustering very reliably generates balanced clusters. This raises the question of what properties a corpus should have so that the clusters are balanced. To this end we introduce a stochastic generative model and rigorously analyze our scheme for data generated by it. Section 2.2 describes the model. The analysis in Section 2.3 begins by establishing several useful mathematical properties of our model. Following these we prove that the model ensures, with high probability, two crucial properties of our simple random clustering technique: (a) the clusters are balanced and (b) the quality of the results found by our technique is near-optimal.

2.1 (Un)balanced Clusters

We first prove that if $d \in \Omega(\log n)$, then there are corpora such that the method is likely to generate unbalanced clusters. We make use of a well-known theorem from information theory. Roughly, the theorem says that, if d is high, then we can find in the d-dimensional hypercube a very large set of points S such that every pair of points from S has large Hamming distance.

THEOREM 2.1. *For each value of n, there is a set of n points with $\Theta(\log n)$ dimensions, for which, with probability $1/e$, the clustering algorithm yields clusters containing $\Theta(n)$ points.*

PROOF. From the Gilbert-Varshamov Theorem (see for instance [24]) it follows that there exists a set $\mathcal{V}$ of $(n-\sqrt{n})$ binary vectors, with $c = \Theta(\log n)$ components, such that

$$\forall x, y \in \mathcal{V} \quad d(x,y) \geq \sqrt{c/3} \tag{4}$$

where d is the Euclidean distance.

Let $\mathcal{C}$ be the set of $\sqrt{n}$ vectors, where each of them has all its components equal to $1/2$. From simple algebra, it follows

$$\forall x \in \mathcal{V}, \forall y \in \mathcal{C} \quad d(x,y) \leq \sqrt{c/4}. \tag{5}$$

Suppose that the clustering algorithm receives in input the set of points in $\mathcal{V} \cup \mathcal{C}$. Let $\mathcal{L}$ be the set of representatives selected by the algorithm. It selects exactly one representative in $\mathcal{C}$ with probability

$$\Pr(|\mathcal{L} \cap \mathcal{C}| = 1) \approx \sqrt{n}\frac{1}{\sqrt{n}}\left(1 - \frac{1}{\sqrt{n}}\right)^{\sqrt{n}-1} \approx \frac{1}{e}. \tag{6}$$

If exactly one representative $r \in \mathcal{C}$ is selected, then all points in $\mathcal{V}$ are attached to the cluster of r. Thus, with constant probability there is a cluster containing $\Theta(n)$ points. □

Remark: The above construction can be extended to the case where all points lie on the unit sphere and the notion of distance is cosine similarity rather than Euclidean distance.

Consider now the following procedure. Select $\sqrt{n}$ random leaders. Produce a new instance by eliminating all points contained in clusters of size $O(\sqrt{n})$. If there are any points left, select $\sqrt{n}$ new leaders, and so on for $O(\log n)$ iterations. The next theorem shows that, if $d = \Omega(\log n)$, there are instances such that at the end of the above procedure there still exist large clusters, i.e. clusters of size $\omega(\sqrt{n}\log n)$. For lack of space, the next theorem is stated without proof.

THEOREM 2.2. *For each value of n, there is a set of n points with $\Theta(\log n)$ dimensions, for which, with probability $1-o(1)$, the clustering algorithm iterated $\log n$ times still yields clusters containing $\omega(\sqrt{n}\log n)$ points.*

In particular, we can construct examples such that there are clusters of size $\Omega(n^{3/4-\epsilon})$, where $\epsilon > 0$. In contrast, if the number of dimensions is constant, the clusters are balanced with high probability.

THEOREM 2.3. *For any set of n points with a constant number d of dimensions, with high probability the clustering algorithm yields clusters containing $O(\sqrt{n}\log n)$ many points.*

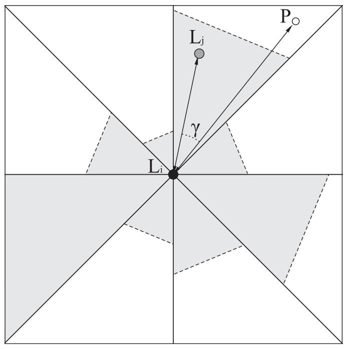

Figure 1: Balanced clusters for constant d

PROOF. Let $\mathcal{L}_1, \ldots, \mathcal{L}_{\sqrt{n}}$ be the representatives selected by the clustering algorithm. We now bound the probability that the number of points attached to a representative $\mathcal{L}_1$ is large. For the sake of simplicity, we assume $\mathcal{L}_1$ to lie at the origin. Because d is constant, it follows that there is a constant k and a partition $C_1, \ldots, C_{k^d}$ of $\mathbb{R}^d$ with the following two properties:

1. the angle between the position vectors of any two points in the same set C_i is at most $\alpha = \pi/4$;
2. $\forall i, \forall j$, there is l, $1 \le l \le k$, such that for all $v \in C_i$ the angle β between v and the jth axis (i.e. the unit vector whose jth component is equal to one) is in the range $2\pi(l-1)/k \le \beta < 2\pi l/k$.

Let D_i be the set of points in C_i that are at distance at most r from $\mathcal{L}_1$, where r is maximal w.r.t. the property that D_i contains no representative.

With probability $1-o(1)$, we have that $|D_i| = O(\sqrt{n}\log n)$ $\forall i = 1, \ldots, k^d$. For each set C_i let $\mathcal{L}_{m_i}$ be the representative in C_i which is closest to $\mathcal{L}_1$. Note that if such a representative does not exist, $|C_i| = O(\sqrt{n}\log n)$.

Because the angle between any two points in C_i is at most α, it follows that any point in $C_i \setminus D_i$ is closer to $\mathcal{L}_{m_i}$ than to $\mathcal{L}_1$, $i = 1, \ldots, k^d$. Thus, the set of points in C_i that are attached to $\mathcal{L}_1$ is a subset of D_i. Since the number k^d of such sets is constant, it follows that with high probability, the cluster of $\mathcal{L}_1$ contains $O(\sqrt{n}\log n)$ points.

By taking the union bound over all representatives we obtain the claimed result. $\square$

Figure 1 shows the construction for two dimensions. The black central point is $\mathcal{L}_i$. The space around it is subdivided into 8 parts. The shaded triangles contain $\sqrt{n}\log n$ points. The grey point is the representative $\mathcal{L}_j$ that, with high probability, exists in a shaded triangle. Since the angle γ between $\mathcal{L}_j$ and P is less or equal than α, the white point P is closer to the grey representative than to the black point.

2.2 The generative model

We now present the *hierarchical mixture of Gaussians* model. We model topics and sub-topics as a hierarchy (tree) of Gaussian distributions. A node u of the tree corresponds to a topic and is associated with the mean vector $\bar{\mu}(u)$ and standard deviation vector $\sigma^2(u)I$ of a multidimensional Gaussian distribution in the vector space. For the root of the tree we use $\bar{\mu}(0)$ and $\sigma^2(0)I$. (I denotes the d-dimensional vector of all 1's.)

1. The number of sub-topics of a non-leaf node is a random integer in the range $[2, \ldots, c]$ for some constant c independent of n.
2. The means of the sub-topics of node u have Gaussian distribution with parameters $(\bar{\mu}(u), \sigma^2(u)I)$.
3. A sub-topic has smaller variance than the parent topic, thus for a sub-topic v of u, $\sigma^2(v) \le (1-\epsilon)\sigma^2(u)$ for some parameter $\epsilon = \epsilon(d)$.
4. The depth of the tree is $h = \alpha \log n$, for some constant $\alpha < 1$.
5. To generate a document we choose a leaf of the tree uniformly at random and generate a point using the distribution associated with this leaf.

Three technicalities of the model are worth mention. First, the separation between sub-topics is controlled by the parameter ϵ. With larger ϵ (smaller variance) documents of a sub-topic are more concentrated near the sub-topic's mean value. With smaller ϵ, documents are scattered in a larger range with less separation between documents corresponding to different sub-topics. Our analysis holds even for very small values of ϵ (we need just $\epsilon = \Omega(\log n/\sqrt{d})$).

Second, the model may generate negative coordinates for some point components; if this is a concern, it can be addressed in a number of ways, such as using only mean vectors with positive components and reflecting about the mean any negative coordinates that survive.

Third, even with all non-negative coordinates, the resulting points are not on the unit sphere (corresponding to normalized document vectors in information retrieval). We prove in Corollary 2.8 that with high probability[2] the vectors generated have almost equal norm, and therefore normalizing the vectors will not change the outcome of the search. Queries are also generated by exactly the same model.

2.3 The analysis

Although our experiments focus on the case $n = O(d)$, we only require a weaker condition for the analysis: that $n < \mathrm{e}^{\gamma d}$ for any constant $\gamma < 1$ (note that once d exceeds about 200, this condition only requires that n is less than the number of atoms estimated to be in the universe). To simplify the presentation and to highlight the novel aspects in the analysis we focus on the case $L = 1$ and assume that the number of sub-topics of each topic is exactly c, and that the d components in each variance vector are the same. (If the number of sub-topics is not the same at each node, we need to augment the analysis below with standard branching process tools. If the variances of different components of the distribution are different, we need to apply a more detailed large deviation bound. However, the end results in terms of the performance of the algorithm are similar.)

2.3.1 *Properties of the Model.*

Let $X = (X_1, \ldots, X_d)$ denote a vector generated by our model. Let $(\bar{\mu}(j), \sigma^2(j)I)$ be the parameters of the distribution of the node at depth j, $0 \le j \le h$, on the path from

[2]Probability at least $1 - 1/d^\theta$ for some constant $\theta > 0$.

the root to the leaf that generated X. Then,

$$X = (X - \bar{\mu}(h)) + \sum_{j=1}^{h}(\bar{\mu}(j) - \bar{\mu}(j-1)) + \bar{\mu}(0),$$

where $(X - \bar{\mu}(h))$ has a Gaussian distribution with parameters $(0I, \sigma^2(h)I)$ and $(\bar{\mu}(j) - \bar{\mu}(j-1))$ has a Gaussian distribution with parameters $(0I, \sigma^2(j-1)I)$. Thus a priori, X has a Gaussian distribution with parameters $(\bar{\mu}(0), \sum_{j=0}^{h}\sigma^2(j)I)$.

Next, assume that X was generated by leaf u_X, Y was generated by leaf u_Y, and that their lowest common ancestor in the tree is u_g at depth g. Suppose that the distribution associated with node u_g has parameters $(\bar{\mu}(g), \sigma^2(g)I)$. Then the distribution of $X - Y$ is Gaussian with parameters $(0I, 2\sum_{j=g}^{h}\sigma^2(j)I)$. Let

$$D(X,Y) = \sqrt{\sum_{i=1}^{d}(X_i - Y_i)^2}$$

be the Euclidean distance between X and Y; then

$$\frac{1}{2(\sum_{j=g}^{h}\sigma^2(j))^2}(D(X,Y))^2$$

has distribution $\chi^2_{(d)}$. We prove a large deviation bound for this distribution.

LEMMA 2.4. *Suppose Z has a distribution $\chi^2_{(d)}$. For any $0 \le \delta \le 1$, we have* $\Pr(|Z - d| \ge \delta d) \le 2e^{-\delta^2 d/3}$.

PROOF.

$$\begin{aligned}\Pr(Z \ge (1+\delta)d) &= \Pr(e^{tZ} \ge e^{t(1+\delta)d}) \le \frac{E[e^{tZ}]}{e^{t(1+\delta)d}} \\ &= \frac{(1-2t)^{-d/2}}{e^{t(1+\delta)d}} \le e^{-\delta t d}.\end{aligned}$$

Setting $t = \delta/3$ we have $\Pr(Z \ge (1+\delta)d) \le e^{-\delta^2 d/3}$. Similar calculations give $\Pr(Z \le (1-\delta)d) \le e^{-\delta^2 d/3}$. □

COROLLARY 2.5. *Let X and Y be two independent random variables with a Gaussian distribution $N(\bar{\mu}, \sigma^2 I)$ in dimension d. For any* $0 \le \delta \le 1$,

$$\Pr(|D^2(X,Y) - 2d\sigma^2| \ge 2\delta d\sigma^2) \le 2e^{-\delta^2 d/3}.$$

Applying the above corollary to our model we get a tight bound on the distances between points in a given topic but different sub-topics.

COROLLARY 2.6. *Suppose that point X was generated by leaf u_X and point Y was generated by leaf u_Y. Suppose that the lowest common ancestor of the two leaves is u_g at depth g. For any $0 < \delta \le 1$,*

$$\Pr(|D^2(X,Y) - 2d(\sum_{j=g}^{h}\sigma_j^2)| \ge \delta 2d(\sum_{j=g}^{h}\sigma_j^2)) \le 2e^{-\delta^2 d/3}.$$

Next we show that the norms of all vectors generated by our model are sufficiently close that normalizing the vectors will not change the search results.

LEMMA 2.7. *Let $X = (X_1, \ldots, X_d)$ be a vector generated by our model. Let $||X|| = \sqrt{\sum_{i=1}^{d} X_i^2}$ be the norm of X. Let $\mathcal{N} = d\sum_{g=0}^{h}\sigma^2(h) + \sum_{i=1}^{d}\mu_i^2(0)$, then for any constant $\delta > 0$*

$$\Pr(|(||X||)^2 - \mathcal{N}| \ge \delta\mathcal{N}) \le 2e^{-\delta^2 d/3}.$$

As a corollary we show that in our model nearest neighbors search with respect to the Euclidean distance function is similar to search with respect to the cosine similarity function

$$C(X,Y) = \frac{X \cdot Y}{\sqrt{\sum_{i=1}^{d} X_i^2}\sqrt{\sum_{i=1}^{d} Y_i^2}}.$$

COROLLARY 2.8. *Let X and Y be two vectors generated by our model. For any $\delta > 0$, with high probability*

$$|(1 - \frac{D^2(X,Y)}{2\mathcal{N}^2}) - C(X,Y)| \le \delta.$$

PROOF.

$$\begin{aligned}D^2(X,Y) &= \sum_{i=1}^{d}(X_i - Y_i)^2 = \sum_{i=1}^{d} X_i^2 + \sum_{i=1}^{d} Y_i^2 - 2\sum_{i=1}^{d} X_i Y_i \\ &= \sum_{i=1}^{d} X_i^2 + \sum_{i=1}^{d} Y_i^2 - 2C(X,Y)\sqrt{\sum_{i=1}^{d} X_i^2 \sum_{i=1}^{d} Y_i^2}.\end{aligned}$$

Applying Lemma 2.7 we have $2\mathcal{N}^2(1-\delta)(1-C(X,Y)) \le D^2(X,Y) \le 2\mathcal{N}^2(1+\delta)(1-C(X,Y))$. □

2.3.2 Quality of the search results.

Consider a query q and let $\ell(q)$ be the closest leader to q. Given that q is also generated by our model, let $u(q)$ be the root of the smallest subtree that includes both q and $\ell(q)$, let $g(q)$ be the depth of $u(q)$, and let $Q(q)$ be the set of points generated by the leaves of this sub-tree.

Applying Corollary 2.6 we have that with high probability, for any pair $\bar{a}, \bar{b} \in Q(q)$

$$D(\bar{a}, \bar{b})^2 \le 2d(\sum_{j=g(q)}^{h}\sigma^2(j))(1+\delta).$$

Similarly, with high probability for any pair $\bar{x} \in Q(q)$ and $\bar{y} \notin Q(q)$

$$D(\bar{x}, \bar{y})^2 \ge 2d(\sum_{j=g(q)-1}^{h}\sigma^2(j))(1-\delta).$$

Using the fact that the variance of the distribution decreases by a factor $(1-\epsilon)$ in each level, and setting $\epsilon > 2\delta/(1+\delta)$ we have that for any pair $\bar{a}, \bar{b} \in Q(q)$ and any pair $\bar{x} \in Q(q)$ and $\bar{y} \notin Q(q)$ with high probability

$$D(\bar{a}, \bar{b})^2 < D(\bar{x}, \bar{y})^2.$$

Thus, if $K \le |Q(q)|$, it suffices to check only points in $Q(q)$, and all points in $Q(q)$ will be in clusters with leaders in $Q(q)$. (Note however that $|Q(q)|$ is a random variable.)

We first estimate the probability that the search returns the exact ground truth set G_q^K. For the practical range of $2 < c < 10$, the following theorem shows that the probability of returning the *exact* ground truth is at least 0.2.

THEOREM 2.9. *If query processing checks the points of only the cluster of the nearest leader ($a = 1$), then*

$$\Pr(A_q^K = G_q^K) \ge \frac{\ln c}{c-1}e^{-\frac{\ln c}{c-1}}.$$

PROOF. **(Sketch)** Assume first that there is a subtree that generated exactly $\sqrt{n}$ points, and q was generated by a leaf of that subtree. The expected number of leaders in this subtree is 1 and, using a Poisson approximation, the probability that the tree has exactly one leader is e^{-1}. We are not guaranteed to have a tree of exactly this size, but since the size of subtrees grows by a factor of c, there must be such a tree in the range $\frac{\ln c}{c-1}\sqrt{n}$ to $\frac{c \ln c}{c-1}\sqrt{n}$. Taking the minimum of the function in this range we get $\frac{\ln c}{c-1}e^{-\frac{\ln c}{c-1}}$. □

We next consider the case when query processing examines more than one cluster ($b > 1$). We show that the probability of not returning the set G_q^K decreases exponentially with b, the number of clusters checked by the algorithm.

THEOREM 2.10. *Suppose that query processing considers all points in the clusters of the b leaders closest to q. Then*

$$\Pr(A_q^K \neq G_q^K) \leq e^{-b/2ec}(1 + e^{-b/2}).$$

Thus, in that case

$$\Pr(CR_K(Rand.Clust.) = 1) \geq 1 - e^{-b/2ec}(1 + e^{-b/2}).$$

PROOF. **(Sketch)** Let $T(q)$ be the subtree that contains the leaf that generated q and has size $b\sqrt{n}/2ec \leq |Q(q)| \leq b\sqrt{n}/2e$. If $T(q)$ has at least one and no more than b leaders then the search finds all the points in G_q^K.

The probability that the set has no leaders is bounded by $e^{-b/2ce}$. The probability that the tree has more than b leaders is bounded (using a large deviation bound for the Poisson distribution) by $e^{-b/2e}e^{-b/2}$. The probability of failure is no more than the sum of these two probabilities. □

Next we analyze the quality of the search result in terms of the CS_K measure.

THEOREM 2.11. *With high probability $CS_K(\mathcal{A}) = 1 - o(1)$.*

PROOF. **(Sketch)** Since the depth of the generative tree is $\alpha \log n$ for some constant $\alpha < 1$, with high probability all leaves generate at least $2K \log^2 n$ data points each. Consider the smallest sub-tree that includes the leaf that generated q and that generated a total of at least $\sqrt{n} \log n$ points. With high probability this tree has at least one and no more than $2 \log n$ leaders.

Consider the leaf u_q that generated q. Out of the $2K \log^2 n$ points generated by this leaf, only $2K \log n$ can belong to clusters with fewer than K points generated by u_q. Since q has the same distribution, with probability $1 - 1/\log n$, the cluster that contains q contains at least K points generated by this leaf, and therefore the distances to q of all K points returned by this search are bounded with high probability by $2d\sigma_0^2(1-\epsilon)^h$. Thus, $Pr(A_q^K = G_q^K) = 1 - o(1)$, and

$$CS_K = \frac{s(A_q^K, q) - worst(q)}{best(q) - worst(q)} \geq 1 - o(1).$$

□

2.3.3 Cluster size

Using an argument similar to the proof of Theorem 2.9 we prove that applying random clustering to input from the generative model gives almost balanced clusters.

THEOREM 2.12. *With high probability all clusters generated by the random clustering algorithm are of size bounded by $O(\sqrt{n} \log n)$.*

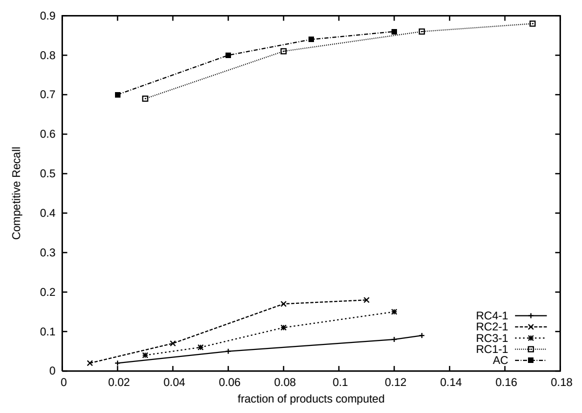

Figure 2: Recall: $L = 1$ works best

3. EXPERIMENTS ON TEXT DOCUMENTS

As mentioned, with synthetic data generated according to the HMG model, our algorithms achieve near-perfect retrieval at a small fraction of the cost. These data are omitted for lack of space and we focus on the experiments with real data. Our data set is a collection of more than 95K scientific papers downloaded from *CiteSeer*. Each document in *CiteSeer* has a URL of the form

http://citeseer.ist.psu.edu/`unique_number`*.html*

We thus downloaded papers by cycling on the `unique_number` that identifies each paper. This process produced an heterogeneous collection, since papers are not grouped by topic (papers with consecutive `unique_number` ID's can deal with completely different topics). When converting documents into unit length vectors, they were stemmed and common stopwords were removed. This process yielded a space with very large dimensionality (over 400K) and with very sparse vectors (average number of non-zero components around 1K).

We also used these documents to generate the queries. Queries of length l were generated by first choosing a document in the collection uniformly at random, and then by selecting (again randomly) a position in the document starting from which l words were read off as the query; although we obtained results similar to the ones below for a wide range of l from 3 to 10,000, the results presented below are for $l = 1000$.

For each choice of L and a in our experiments, five different random clusterings were produced. We refer to these five as (L, a)*-instances.* Each data point in the graphs shown in the following figures is the average taken over 200 *runs*, each as follows: for fixed L and a, a query of the desired length was generated as described above. The query was answered by choosing an (L, a)-instance at random and running the query processing step. During query processing we study the quality of retrieved results as a function of computation. To vary the amount of computation, we looked first at the cluster of the leader closest to the query, then the second closest, etc. Thus by varying this number of clusters (the parameter b), we could vary the computational effort.

Some of the most important conclusions of our study are summarized in Figure 2. On the x-axis we have the com-

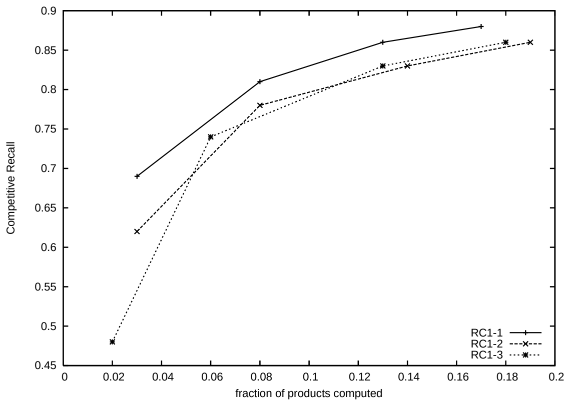

Figure 3: $a = 1$ works best

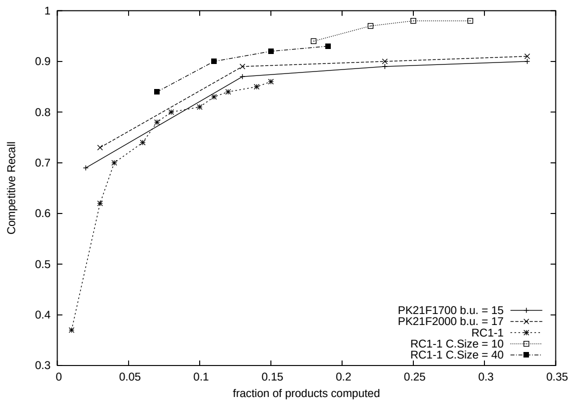

Figure 4: Competitive recall: R. C. vs. p-spheres

putational effort expressed as a fraction of the number of multiplications used to compute the ground truth by measuring all distances, for $K = 30$. On the y-axis we report the competitive recall (Equation (1)). Two curves stand out, with competitive recall higher than 80% for only a tenth of the computational effort. These two curves are for random clustering with $L = 1$, and agglomerative clustering. The figure shows RCL-a for different values of the parameters; for instance, RC2-1 refers to random pruning with 2 levels of recursion and where each point attaches to the closest leader. (The parameter b implicitly varies on the x-axis.) The plot shows that more levels of recursion result in a dramatic deterioration of performance. This is attributable to the fact that as L increases, the clusters at the higher levels of the recursion tree become larger, and so do not have informative leaders. The plot also shows that agglomerative clustering and random clustering deliver essentially similar retrieval quality at any fixed computational effort. Given the former's considerably higher preprocessing cost (at least $\Omega(n^2)$ distance computations), this suggests that for cluster pruning, more complicated clustering is not worth it.

Results for normalized competitive similarity which, as argued, is a better proxy for user perception than competitive recall, reinforce the main conclusions. The simplest scheme, with less than 10% of the computational effort of naive retrieval yields over 90% of the competitive recall (data omitted for lack of space).

The second set of plots, shown in Figure 3, investigates the role of a, the number of representatives each data point attaches to during preprocessing. Again, K=30. Values of a larger than 1 show a marked deterioration. This is because with larger clusters leaders become less discriminating.

A robust conclusion of our study is that centroids are the best leaders. Medioids work reasonably well, while using representatives as leaders consistently gives poor results. We omit the detailed plots in this abstract.

The data so far concerned long queries. Qualitatively the results still hold for short queries even though, not surprisingly, there is some deterioration of performance. In any case, spending about 15% of the computation time of the linear scan still attains competitive recall of about 70% (data omitted for lack of space).

The evidence so far strongly supports the claim that the most effective scheme is obtained when $L = 1$ (one set of leaders, no recursion) and $a = 1$ (every point attaches to only one cluster). This conclusion is robust with respect to in-memory applications, for which the measures of computational cost seen so far are an accurate proxy. Intuitively, on the other hand, some redundancy (i.e., $a > 1$) should be helpful for disk applications, where we want to minimize the number of disk seeks. We study this next.

We now report on a comparison between our scheme and p-spheres [25]. Our implementation followed the specifications given in the original paper for the best performing version (ND Pk-Sphere Trees) [25]. In what follows we show a representative sample of our experiments. The conclusions that we draw are robust, in the sense that they were confirmed by all our experiments.

The p-spheres' fanout, corresponding to the number of leaders of our simple scheme, was varied between 200 and 2000 (step 300), corresponding (roughly) to between half and six times the number of leaders of our simple scheme, respectively. The choices of K, that for p-spheres must be decided at indexing time, varied from 1 to 41 (step 20). To generate the p-spheres we used a random sample of the true query set, later used for the test.

We begin with the in-memory scenario. Figure 4 reports the competitive recall (y-axis) plotted against the computational effort (x-axis). We selected the best performing instances of p-spheres, regardless of space blowup which, as we will see, is very severe. For random clustering we selected the simplest, best performing scheme ($L = a = 1$) varying the cluster size. The plot shows that random clustering is somewhat superior: for the same computational effort it attains better quality. Note however, that while random clustering is space optimal the blowup for p-spheres is significant (15 and 17 in Figure 4). Results for average cosine similarity (omitted for lack of space) confirm this fully. According to the choice of parameters, the space blow-up of p-spheres with respect to the data set size varies from 15 to 130. In spite of this high space consumption, a good 10% of the data set is oftentimes not indexed at all, and hence unretrievable. We now turn to external memory applications. Figure 5 plots quality, measured as competitive recall, against disk I/O time, expressed in milliseconds. We used a high-end disk with average random seek time = $3.6ms$ and Sequential Transfer Rate = $96MB/s$. For comparison, keep in mind that the disk I/O time of the naive linear scan is (roughly)

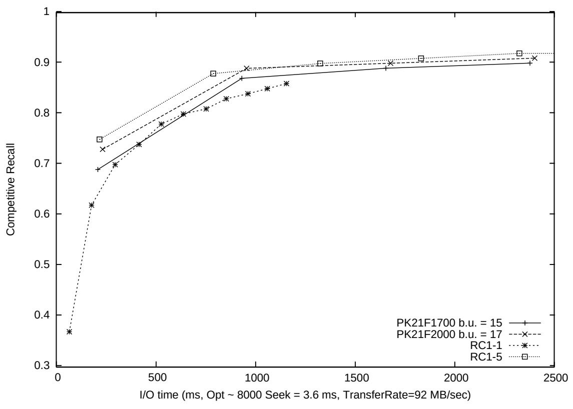

Figure 5: I/O Time: Rand.Clust. vs. p-spheres

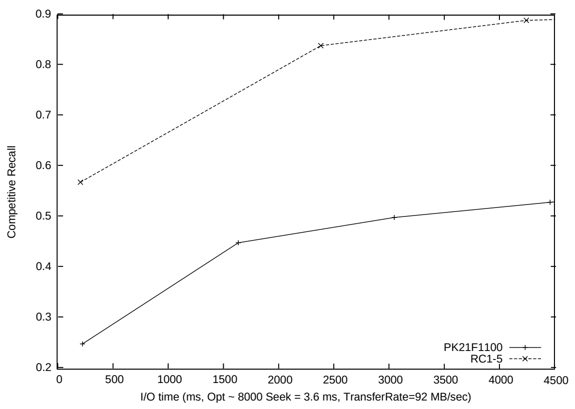

Figure 6: Nasty Queries

$8,000ms$. Again, of the many instances of p-spheres we selected the best performing. For random clustering we show the results for L=1, a=1 and L=1, a=5. Intuitively while for in-memory applications we can improve precision by increasing the number of clusters probed b, doing so in disk applications we pay dearly, for each new cluster considered incurs a seek. Thus, a way to improve precision is by making documents belong to several clusters. The plot shows that in terms of the quality vs. time tradeoff the two schemes are fairly comparable, but while the space blowup of random clustering is only 5, that of p-spheres is almost 20. Whereas the space blowup of p-spheres is an unpredictable function of the various parameters of the pre-processing stage, with random clustering it coincides with the value of a. While a=1 is the best choice for in-memory applications, a higher value of a gives better results for disk I/O settings. The best quality vs time trade-off was for a=5. Larger values of a (up to 20) do not seem to improve it.

We conclude with an important experiment. Recall that p-spheres require the query distribution to be known in advance, while no such prior knowledge is needed by random clustering. Figure 6 shows that p-sphere performance deteriorates significantly if the query distribution used in the preprocessing stage is not accurately known, while random clustering is much more robust in this respect. Recall that at the end of the pre-processing stage there might be documents not belonging to any p-sphere. To produce this last plot, we used these documents as queries. While this query distribution is to some extent artificial, the point is that the sample used to train p-spheres might not be an accurate depiction of reality. Moreover, the set of queries that users submit may change over the period of time, making the initial sample obsolete. In such a case, p-sphere performance may suffer a severe deterioration.

5. REFERENCES

[1] S. Arya, D. M. Mount, N. S. Netanyahu, R. Silverman, and A. Wu. An optimal algorithm for nearest neighbor searching. In *SODA'94*.

[2] S. Berchtold, K. Keim, and H.-P. Kriegel. The X-Tree: An index structure for high dimensional data. In *VLDB'96*.

[3] E. Kushilevitz, R. Ostrovsky, and Y. Rabani. Efficient search for approximate nearest neighbor in high dimensional spaces. *SIAM Journal on Computing*, 30(2):451-474, 2000.

[4] M. Bern. Approximate closest point queries in high dimensions. *Information Processing Letters*, 45, 1993.

[5] T Bozkaya and M. Ozsoyoglu. Distance-based indexing for high-dimensional metric spaces. In *PODS'97*.

[6] K. Clarkson. Nearest neighbor queries in metric spaces. In *STOC'97*.

[7] R. Motwani, P. Indyk. Approximate nearest neighbor - towards removing the curse of dimensionality. In *STOC'98*.

[8] Vladimir Pestov. On the geometry of similarity search: dimensionality curse and concentration of measure. *Information Processing Letters*, To Appear.

[9] K. S. Beyer, J. Goldstein, R. Ramakrishnan, and Uri Shaft. When is "nearest neighbor" meaningful? In *ICDT '99*.

[10] K. P. Bennett, U. Fayyad, and D. Geiger. Density-based indexing for approximate nearest-neighbor queries. In *KDD '99*.

[11] Sergey Brin. Near neighbor search in large metric spaces. In *The VLDB Journal*, 574–584, 1995.

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

[13] A. Gionis, P. Indyk, and R. Motwani. Similarity search in high dimensions via hashing. In *The VLDB Journal*, 1999.

[14] H. Edelsbrunner. *Algorithms in Combinatorial Geometry*. Springer-Verlag, New York, NY, 1987.

[15] L. Ertz, M. Steinbach, and V. Kumar. Finding topics in collections of documents: A shared nearest neighbor approach. In *Text Mine '01*.

[16] C. Buckley and A. F. Lewit. Optimization of inverted vector searches. In *SIGIR '85*, 97–110.

[17] M. Steinbach, G. Karypis, and V. Kumar. A comparison of document clustering techniques. In *KDD Workshop on Text Mining, 2000*.

[18] P. Willet. Recent trends in hierarchical document clustering: a critical review. In *Information Processing and Management*, vol. 24(5), 577-597, 1988.

[19] D. Comer. The ubiquitous b-tree. In *ACM Computing Surveys, 11(2):121137, 1979*.

[20] A. Guttman. R-trees: A dynamic index structure for spatial searching. In *SIGMOD '84* .

[21] G. Karypis, E-H Han, and V. Kumar. Chameleon: Hierarchical Clustering Using Dynamic Modeling. IEEE *Computer*, 32(8):68–75, August 1999.

[22] N. Katayama and S. Satoh. The sr-tree: An index structure for high-dimensional nearest neighbor queries. In *SIGMOD'97* .

[23] A. N. Papadopoulos Y. Manolopoulos, A. Nanopoulos and Y. Theodoridis. R-trees have grown everywhere. In *Technical Report available at http://www.rtreeportal.org/*, 2003.

[24] F. J. MacWilliams and N. J. A. Sloane. The Theory of Error Correcting Codes. Amsterdam: North-Holland, 1977.

[25] J. Goldstein and Raghu Ramakrishnan. Contrast Plots and P-Sphere Trees: Space vs. Time in Nearest Neighbour Searches. In *VLDB'00*.