Indexing inexact proximity search with distance regression in pivot space

Ole Edsberg
edsberg@idi.ntnu.no

Magnus Lie Hetland
mlh@idi.ntnu.no

Department of Computer and Information Science
Norwegian University of Science and Technology
Trondheim, Norway

ABSTRACT

We introduce an inexact indexing scheme where, at index building time, training queries drawn from the database are used to fit one linear regression model for each object to be indexed. The response variable is the distance from the object to the query. The predictor variables are the distances from the query to each of a set of pivot objects. At search time, the models can provide distance estimates or probabilities of inclusion in the correct result, either of which can be used to rank the objects for an inexact search where the true distances are calculated in the resulting order, up to a halting point. To reduce storage requirements, the coefficients can be discretized at the cost of some precision in the promise values. We evaluate our scheme on synthetic and real-world data and compare it to a permutation-based scheme that has been reported to outperform other methods in the same experimental setting. We find that, in several of our experiments, the regression-based distance estimates give better query performance than the permutation-based promise values, in some cases even when the pivot set for the regression-based scheme is reduced in order to make its memory size equal to that of the permutation-based index. Limitations of our scheme include high index building cost and vulnerability to deviation from the model assumptions.

Categories and Subject Descriptors

H.3 [**Information Storage and Retrieval**]:
Content Analysis and Indexing—*Indexing Methods*

General Terms

Algorithms, Performance

1. INTRODUCTION

Proximity search has a variety of applications, including k nearest neighbours classification and multimedia information retrieval. In this paper, we introduce an indexing scheme to facilitate faster proximity search, with the restriction that the indexes may not access the internal structure of the objects to be indexed, only the output of the distance function. This restriction makes the indexing scheme more generally applicable, and places it in the category *distance-based indexing*. This category encompasses both *metric indexing* [5, 8, 9], where the distance function is assumed to satisfy the metric properties, and *non-metric indexing* [16], where it is not. Our indexing scheme is applicable both to metric and non-metric spaces, and to both of the main query types: k-NN queries, which must return the k objects with the smallest distance to the query object, and range queries, which must return all objects whose distances to the query object do not exceed a radius r.

Indexes that are guaranteed to return the correct result every time without error are called *exact*, whereas those without such a guarantee are called *inexact*. Some inexact indexing schemes are approximate, meaning that they provide a guarantee of how wrong the returned objects can be, some are probabilistic, meaning that they provide a guarantee of the probability that the result will be correct, some are probabilistically approximate, and some provide no guarantee at all [13]. Our regression-based indexing scheme, like the permutation-based scheme we will compare it against, is inexact and provides no guarantees. Its performance must be assessed empirically.

The *Curse of Dimensionality* is a phenomenon that affects many types of distance-based applications, including indexing of proximity search. It manifests in dramatic performance degradation as data dimensionality increases. The curse is not triggered by tuple size in itself (which may not even apply for some data sets), but by the *intrinsic* dimensionality, which depends on the distance distribution [5]. For high enough dimensionality, exact indexes fail completely and must compute the distance from the query to every indexed object. If exactness is not an absolute requirement, it may still be possible to obtain a mostly correct answer without calculating more than a fraction of the possible distances by making a trade-off between search accuracy and computational cost [4]. The curse is dampened, not cancelled; as the intrinsic dimensionality increases, we will have to settle for worse computation/accuracy trade-offs.

When the distance function is expensive to compute, a simple way to perform an inexact proximity search with a trade-off between computation and accuracy is to visit the objects in the database in the order, or schedule, given by a cheaply computed *promise value function*, calculating the true distance for each object only as it is visited [3]. The objects after the halting point of the schedule are eliminated without calculating the true distance, incurring a risk of losing members of the true query result but also a saving some distance computations. By adjusting the halting point, we can obtain different computation/accuracy trade-offs. The promise value function may require a precalculated data structure, which constitutes an index for the database. When designing indexing schemes in this setting, a central problem is to choose a promise value function that will place the members of the true query result as early as possible in the ordering.

In this article, we introduce a new indexing scheme where distance regression is used to obtain promise values for inexact proximity search. We fit one regression model for each object to be indexed. We fit the models on training queries drawn from the database, with the true distance from the indexed object to the query as the response variable, and the true distances from the indexed object to a set of pivot objects as the predictor variables. We fit one model for each object to be indexed. The models can produce both distance estimates and inferences about the probability of the indexed object being part of the true query result. With n indexed objects and m pivots, the scheme requires the storage of $n(m+1)$ coefficients for the production of distance estimates, with an additional $2n+2$ values required for the probabilistic inferences. The coefficients can be discretized, at the cost of some precision, to reduce the memory size of the index. Building our indexes is computationally expensive: Distances between the n database objects and the n' training queries must be calculated, and n regression models must be fitted. We compare the performance of our scheme against the permutation-based indexing scheme proposed by Chávez et al. [3], which was reported to perform better than other methods in the same setting. We reuse data sets, plot types and parameters from their study in our evaluation. In several of our experiments, our distance estimates give better query performance than the permutation-based promise values, in some cases even when the pivot set for the distance estimates is reduced in order to equalize the memory size of the indexes.

2. TERMINOLOGY

Given a database $\mathbb{S} = \{u_1, u_2, \ldots, u_n\}$, which is a finite subset of a universe $\mathbb{U}$ of objects, and a distance function $d : \mathbb{U} \times \mathbb{U} \to \mathbb{R}$, our task is to facilitate fast proximity search (of the range and/or k-NN type) for queries drawn from $\mathbb{U}$.

The indexes we consider operate with reference to an ordered set of pivot objects $\mathbb{P} = (p_1, p_2, \ldots, p_k) \in \mathbb{U} \setminus \mathbb{S}$.[1] $\mathbb{P}$ gives rise to a vector space embedding $\Phi(o) : \mathbb{U} \to \mathbb{R}^k$, mapping each object $o \in \mathbb{U}$ to $(d(o,p_1), d(o,p_2), \ldots, d(o,p_k))$.

Search is performed by calculating $\Phi(q)$ for the query object q, and then visiting $\mathbb{S}$ in the ordering,[2] or schedule, given by a promise value function $\mathfrak{P}(u, \phi, r) : \mathbb{S} \times \mathbb{R}^k \times \mathbb{R} \to \mathbb{R}$. The parameter r in $\mathfrak{P}(u, \Phi(q), r)$ stands for the radius of a range query, and is only needed for probability-based promise values. For k-NN queries it can be estimated from $\mathbb{S}$. At each point u in the schedule, the true $d(u,q)$ is calculated. By halting the search before the schedule has been completed, we save some distance calculations at the cost of possibly discarding some members of the true query results.

[1]For normal operation of the indexes, $\mathbb{P}$ would be taken from $\mathbb{S}$. Our interest is to compare the schedules produced by different promise value functions. For this purpose, having $\mathbb{P}$ as part of $\mathbb{S}$ would be a distraction because $\mathbb{P}$ in effect would be prepended to every schedule.

[2]To order the objects, we used the incremental sort algorithm of Paredes & Navarro [12].

3. RELATED WORK

Indexing methods for inexact similarity search have been surveyed by Patella & Ciaccia [13]. Of particular relevance to our regression-based scheme is a line of work [2, 4] starting with the idea of shrinking the query radius by a multiplicative factor, sacrificing exactness in return for increased filtering power, especially in spaces with high intrinsic dimensionality. This idea was applied to a simple pivot-based LAESA-like [11] indexing scheme where the lower bound from the triangle inequality was used to filter the database. Instead of controlling the computation/accuracy trade-off by shrinking the radius, it can, as done by Chávez et al. [3], be controlled by sorting the database according to the lower bound and adjusting the size of the fraction (with the lowest bounds) of objects that are not filtered away. This is the promise value based search mechanism we use in this paper.

Using the promise value based search mechanism, Chávez et al. introduced a permutation-based indexing scheme, where the promise value function was the rank correlation (Spearman's rho) between two permutations of the pivots: the permutation according to distance to the query object and the permutation according to distance to the indexed object [3]. They found that this scheme outperformed several other schemes, including promise values based on the L_1, L_2 and L_∞ norms between $\Phi(q)$ and $\Phi(u)$. We will apply their scheme as a baseline in our experiments, and we label it $\mathfrak{P}_\Pi$.

BoostMap [1] is another index that performs promise value based search, and that operates with reference to a set of (candidate) pivot objects. At index-building time, BoostMap applies an AdaBoost-like algorithm to construct a mapping from objects in $\mathbb{U}$ to vector space embeddings. These embeddings are linear combinations of a number of one-dimensional embeddings, selected by the algorithm from a large number of randomly generated candidates. The 1D embeddings come in two types: one type embeds an object as its distance to a single pivot object, and the other type embeds an object as its projection onto the line between a pair of pivot objects. The algorithm is optimized with reference to a set of triplets of objects, and tries to maximize the degree to which the distances between embedded objects agree with the true distances on the question of whether the second or the third object in a triplet is closer to the first. Only triplets where the second object is a k-NN of the first (for k up to a parameter $k_{\max}$) are included, thus focusing the optimization on preserving the nearest-neighbour structure in the data set.

4. INDEXING INEXACT PROXIMITY SEARCH WITH DISTANCE REGRESSION IN PIVOT SPACE

A good promise value function is one that, among other criteria, (i) places the true query results early in the schedule, thus providing good computation/accuracy trade-offs, (ii) can be calculated quickly enough relative to the true distance, and (iii) requires little space for its index. Our regression-based indexing scheme is motivated by criterion (i) and stems from two lines of thought:

1. *The promise value ordering should approximate as closely as possible the ordering of the indexed objects according to their true distance to the query (ascending).* This line of thought undoubtedly lies behind the permutation based indexing scheme of Chávez et al. [3], with the reasoning that objects with a higher pivot permutation correlation to the query are more likely to have a smaller true distance to it. We take a more direct route, and instead reason that objects with a smaller estimated distance to the query are more likely to have a smaller true distance to it. We will show how linear regression models can provide the needed distance estimates. We call the resulting promise value functions $\mathfrak{P}_{\hat{d}}$.

2. *The promise value ordering should approximate as closely as possible the ordering of the indexed objects according to their probability of being part of the true query result (descending).* This will, for any given halting point in the search schedule, maximize the expected number of members of the true query result that will have been found at that point. The application of any distance-based index to a query object can in this way be viewed (explicitly or implicitly) as a problem of probabilistic inference based on the incomplete information stored in the index, as well as the query parameters and the distance calculations performed as part of the search. The probability-based promise values take into account that the reliability of the estimates may vary between the indexed objects, for example, as a consequence of their position in relation to the pivot objects. We will show how linear regression models can provide such inferences for use as promise values. We call the resulting promise value functions $\mathfrak{P}_t$.

To see how ordering according to probabilities could give a different outcome from ordering according to estimated distances, consider a situation where two indexed objects a and b are both estimated to fall outside the query radius, and where the estimated distance from the query to a is just a little bit smaller than that to object b. Assume additionally that the index is known to produce much more reliable distance estimates for a than for b. Then it may be the case that, given the information available to us, b has a higher probability of belonging to the query result than a, even though a has a smaller estimated distance to the query.

While the probability-based promise values are justified in terms of maximizing search accuracy, it is not a given that they will perform better than the distance-based ones in practice; for example, the inference may depend on assumptions that are not fully satisfied by the distance space.[3]

[3]In our experiments, $\mathfrak{P}_t$ never performs better than $\mathfrak{P}_{\hat{d}}$. We still include $\mathfrak{P}_t$ because of its interesting rationale.

4.1 The linear models

We apply linear regression [6] to model the relationship between $d(u,q)$ and $\Phi(q)$, for each $u \in \mathbb{S}$. We fit n models, one for each u. Given a query q, the model for u allows us to estimate $d(u,q)$, and also to make inferences about $\Pr(d(u,q) \leq r)$.

Let $\Phi^1(o)$ be the vector resulting from prepending 1 to $\Phi(o)$. We assume that the distance from an indexed object u to a query q is equal to the dot product between $\Phi^1(q)$ and $m+1$ coefficients β_u plus an additive error term ϵ_u. The errors are assumed to be i.i.d. with mean 0. We can write the model for u as:

$$d(u,q) = (\Phi^1(q))^T \beta_u + \epsilon_u \tag{1}$$

To estimate β_u, we select, based on an assumption that $\mathbb{S}$ is representative of the queries the index will be applied to, a random sample $\mathbb{S}' \subseteq \mathbb{S}$ of n' training queries. We calculate $\Phi^1(q')$ for each $q' \in \mathbb{S}'$. When fitting the model that will index u, we exclude u from $\mathbb{S}'$ to avoid using a training query with $d(u,q') = 0$. We denote the training queries that will be used to index u as $q'_{\langle u,1\rangle}, q'_{\langle u,2\rangle}, \ldots, q'_{\langle u,n'_u\rangle} = \mathbb{S}' \setminus u$. From these, we construct the n'_u-length vector $\mathbf{y}_u$ and the $n'_u \times (m+1)$ matrix $\mathbf{X}_u$:

$$\mathbf{y}_u = \begin{pmatrix} d(u, q'_{\langle u,1\rangle}) \\ d(u, q'_{\langle u,2\rangle}) \\ \cdots \\ d(u, q'_{\langle u,n'_u\rangle}) \end{pmatrix} \quad \mathbf{X}_u = \begin{pmatrix} \Phi^1(q'_{\langle u,1\rangle}) \\ \Phi^1(q'_{\langle u,2\rangle}) \\ \cdots \\ \Phi^1(q'_{\langle u,n'_u\rangle}) \end{pmatrix}$$

Using $\mathbf{y}_u$ and $\mathbf{X}_u$, we obtain a least squares estimate $\hat{\beta}_u$ of β_u via QR decomposition (using the `lm.fit` function in R [15]), by solving the following linear regression problem:

$$\mathbf{y}_u = \mathbf{X}_u \beta_u + \boldsymbol{\epsilon}_u \tag{2}$$

Deviation from the assumptions may degrade the accuracy of estimates and inferences based on the model [6]. Since the search is inexact to begin with, this can be acceptable. In any case, empirical evaluation will show how well an index performs in practice.

4.2 Calculating promise values

The model for the indexed object u provides a distance estimate that can be used as a promise value for u with respect to the query q:

$$\mathfrak{P}_{\hat{d}}(u, \Phi(q), r) = \hat{d}(u,q) = \Phi^1(q)\hat{\beta}_u \tag{3}$$

We have already assumed that the distance conforms to Equation 1, with the errors i.i.d. and having mean 0. To make the probabilistic inferences we need for $\mathfrak{P}_t$, we must make the further assumption that the errors are normally distributed. The variance of the distance estimate is:

$$\mathrm{Var}(\hat{d}(u,q)) = (\Phi^1(q))^T (\mathbf{X}_u^T \mathbf{X}_u)^{-1} \Phi^1(q) \sigma_u^2 \,, \tag{4}$$

where $\sigma_u^2 = \mathrm{Var}(\epsilon_u)$. We can estimate σ_u^2 with $\hat{\sigma}_u^2$ as follows:

$$\hat{\sigma}_u^2 = \frac{\sum_{i=1}^{n'_u} (d(u, q'_{\langle u,i\rangle}) - \hat{d}(u, q'_{\langle u,i\rangle}))^2}{n'_u - m - 1} \tag{5}$$

The following statistic is t-distributed with $n'_u - m - 1$ degrees of freedom:

$$T = \frac{d(u,q) - \hat{d}(u,q)}{\hat{\sigma}_u \sqrt{1 + (\Phi^1(q))^T (\mathbf{X}_u^T \mathbf{X}_u)^{-1} \Phi^1(q)}} \tag{6}$$

If the assumptions of the model hold, the probability that u is part of the true query result is

$$\Pr(d(u,q) \leq r) = \mathrm{PT}\left(\frac{r - \hat{d}(u,q)}{\hat{\sigma}_u\sqrt{1 + (\Phi^1(q))^T(\mathbf{X}_u^T\mathbf{X}_u)^{-1}\Phi^1(q)}}\right), \quad (7)$$

where PT is the cumulative t-distribution with $n'_u - m - 1$ degrees of freedom. If $\mathbf{X}_u$ were the same for all $u \in \mathbb{S}$, the following promise value function would sort $\mathbb{S}$ according to $\Pr(d(u,\Phi(q)) \leq r)$:

$$\mathfrak{P}_t(u,\Phi(q),r) = \frac{r - \hat{d}(u,q)}{\hat{\sigma}_u} \quad (8)$$

In our case, however, recall that when we fit the model for u, we exclude u from the $\mathbb{S}'$. Therefore $\mathbf{X}_u$ will have one row less for those u that are selected to be part of $\mathbb{S}'$. We choose to ignore these differences, and use Equation 8 to calculate the probabilistic promise values.

Because $\mathfrak{P}_t$ makes use of the query radius r, it cannot be applied directly to k-NN queries. However, for each $k \in 1 \ldots n$, we can calculate the radius r that would return on average exactly k of the indexed objects as results for the training queries we have already sampled. We store these radii as part of the index.

4.3 Storage requirements

Calculating $\mathfrak{P}_{\hat{d}}$ for n indexed objects with m pivots requires the storage of $n(m+1)$ coefficients. However, because the promise value function is only used to order $\mathbb{S}$, and because $\Phi^1(q)$ is the same for all $u \in \mathbb{S}$, applying the same linear transform to all $\hat{\beta}_u$ will not change the promise value orderings. Let us denote the highest and lowest value among all coefficients for all models as $\hat{\beta}^+$ and $\hat{\beta}^-$, respectively. A simple way to discretize the coefficient $\hat{\beta}_{\langle u,i\rangle}$ to a p bit representation is to replace it by

$$\left\lfloor \min\left\{\frac{\hat{\beta}_{\langle u,i\rangle} - \hat{\beta}^-}{\hat{\beta}^+ - \hat{\beta}^-} \times 2^p, 2^{p-1}\right\}\right\rfloor. \quad (9)$$

Discretization reduces the precision of the coefficients, and this inaccuracy will carry over to the promise values and possibly degrade query performance.

To calculate $\mathfrak{P}_t$, we additionally need to store the denominator of Equation 8 for each u. If we discretize the model coefficients, we need to store $\hat{\beta}^+$ and $\hat{\beta}^-$, and incorporate these and $\Phi^1(q)$ into Equation 8 to scale r properly. To support k-NN queries with $\mathfrak{P}_t$, we need to store the n estimated radii. We do not discretize the additional $2n+2$ values, as they are dominated by the coefficients in terms of size.

4.4 Computational cost of building the index

Building a regression based index for n objects with m pivots and n' training queries requires $n'(n+m)$ distance calculations and the solution of n linear regression problems, each with matrix size $n' \times (m+1)$ or $(n'-1) \times (m+1)$. In comparison, the permutation based method requires only nm distance calculations.

5. EXPERIMENTAL EVALUATION

To reduce the number of arbitrary choices we make in our evaluation, we imitate the setup used by Chávez et al. [3] as far as possible, given the data sets available on the SISAP site.[4] Following their example, we select the pivots randomly. Because the object of study is the effectiveness of the different schedules, we keep the pivots separate from the indexed objects. We summarize the results in plots with a measure of computational cost on one axis and a measure of query accuracy on the other axis. We measure computation cost with the wall clock time and/or the percentage of the database having been compared to the query with a distance computation at that point in the schedule. In all but one case, we measure query accuracy as the percentage of the true query result having been retrieved at that point. As a baseline for comparison, we use $\mathfrak{P}_\Pi$, the permutation-based indexing scheme found by Chávez et al. [3] to generally outperform the other methods they had selected.

Indexing involves a trade-off between the memory size of the index and the query speedup it offers. To index n objects with m pivots, $\mathfrak{P}_\Pi$ requires $nm\lceil\log_2(m)\rceil$ bits of memory to store its index [3]. The indexes for $\mathfrak{P}_{\hat{d}}$ require $n(m+1)p$ bits, where p is the number of bits used to store a single coefficient. To make a fair performance comparison between $\mathfrak{P}_\Pi$ and $\mathfrak{P}_{\hat{d}}$, we reduce the number of pivots available to $\mathfrak{P}_{\hat{d}}$ by an amount that equalizes the memory size of the two indexes. If $\mathfrak{P}_\Pi$ uses m pivots, a version of $\mathfrak{P}_{\hat{d}}$ using p bits per pivot can only use $\lfloor \frac{m\lceil\log_2 m\rceil}{p} \rfloor - 1$ pivots if we want the comparison to be fair to $\mathfrak{P}_\Pi$.[5] We include both fair and unfair versions of $\mathfrak{P}_{\hat{d}}$ in the evaluation, and mark the fair ones with asterisks. We evaluate four versions of our regression-based indexing scheme:

$\mathfrak{P}_{\langle \hat{d}, 16\,\mathrm{b}\rangle}$ uses $\hat{d}(u,q)$ as promise values, with the coefficients discretized to 16 bits. For 256 pivots, it uses twice the memory of $\mathfrak{P}_\Pi$.

$\mathfrak{P}^*_{\langle \hat{d}, 16\,\mathrm{b}\rangle}$ uses $\hat{d}(u,q)$ as promise values, with the coefficients discretized to 16 bits and with a reduced pivot set to memory-equalize it with $\mathfrak{P}_\Pi$.

$\mathfrak{P}^*_{\langle \hat{d}, 12\,\mathrm{b}\rangle}$ is identical to $\mathfrak{P}^*_{\langle \hat{d}, 16\,\mathrm{b}\rangle}$, except that it only uses 12 bits for each coefficient. This means that it can keep more pivots after memory-equalization, but may suffer a greater loss of precision from the discretization.

$\mathfrak{P}_{\langle t, 16\,\mathrm{b}\rangle}$ uses promise values from Equation 8, with the coefficients discretized to 16 bits. For 256 pivots, it uses slightly more than twice the memory of $\mathfrak{P}_\Pi$.

We programmed the index building in R [15], with query execution and distance calculation written in C. We used $n' = 2000$ training queries, except in the one case where the database was smaller than that. Except where otherwise noted, we averaged performance measures over 1000 queries with $q \notin \mathbb{S}$.

[4]`http://sisap.org`

[5]Note that we do not include the cost of calculating $\Phi(q)$ in the plots of our results, even though the memory-equalized methods will spend slightly less computation time here because they use fewer pivots.

5.1 Experiments on synthetic data

Unit cube with L_2 distance. Fig. 1 shows range query performance on L_2 spaces with 10 000 unit cube vectors with 128, 256, 512 and 1024 dimensions, and with 128 and 256 pivots, with radius set to capture 0.05 % of the objects on average. We measured percentage of the true result retrieved vs. the percentage of distance calculations required.

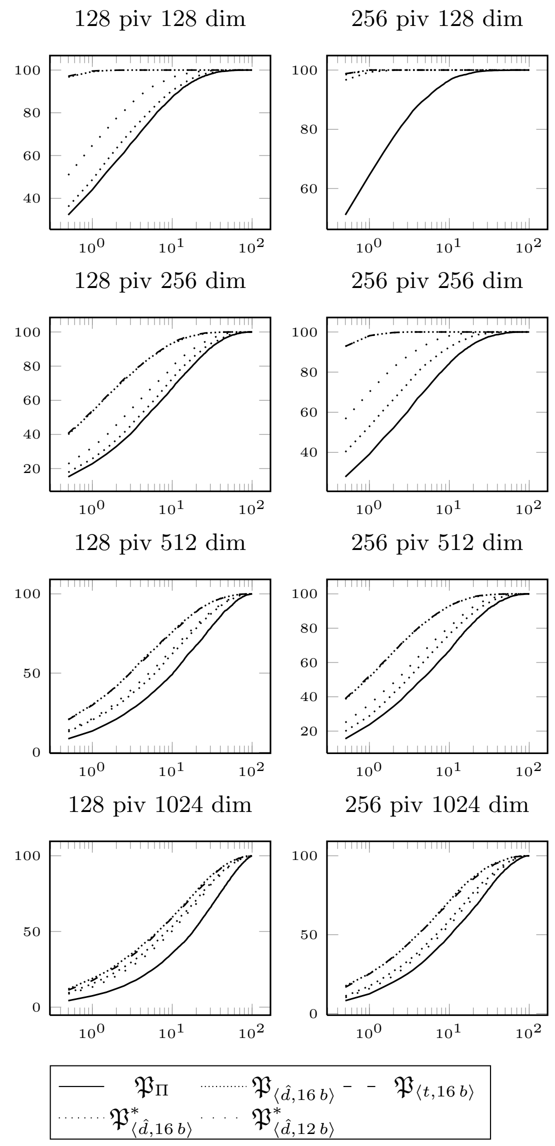

Figure 1: Index performance on range queries on L_2 distance between uniformly distributed vectors, with radius set to capture 0.05 % of the objects. On y axis: avg. % of true query result retrieved. On x axis: % of max distance calculations.

Clustered vectors with L_2 distance. Fig. 2 shows the performance of 1-NN queries on L_2 spaces with 10 000 1024-dimensional vectors generated from a uniform mixture of 32 multivariate normal distribution with variance 0.01 and the means for the distributions generated with another multivariate normal distribution with mean 0.5 and variance 0.09. We used 32 and 128 pivots.

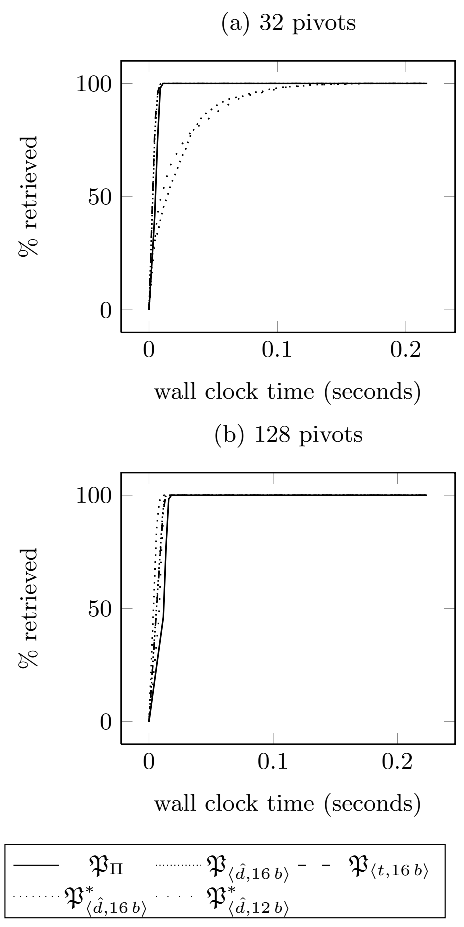

Figure 2: Index performance on 1-NN queries on L_2 distance between vectors generated from a mixture of 32 multivariate normal distributions.

Unit cube with fractional L_p distance (non-metric). Fig. 3 shows index performance on 2-NN queries on $L_{0.2}$ and $L_{0.8}$ spaces, with 10 000 uniformly generated vectors with 32, 64 and 128 dimensions, and with 128 and 256 pivots. Because of a coefficient discretization issue explained in section 5.3, we also included $\mathfrak{P}_{\hat{d}}$ with no discretization (using 8 bytes per coefficient) in this plot, and labelled it $\mathfrak{P}_{\langle \hat{d},\text{nodiscr}\rangle}$.

Index building time. We timed index building for a unit cube L_2 space with 1024 dimensions and 10 000 objects. We used 256 pivots. $\mathfrak{P}_\Pi$ spent about 50 seconds, and $\mathfrak{P}_{\langle \hat{d},16\,\text{b}\rangle}$ spent about 1900 seconds.

5.2 Experiments on real-world data

Face images (FERET) with L_2 distance. Fig. 4 shows performance of k-NN queries, with k ranging from 1 to 20, on images from FERET [14], a database of face images. We used the 761-coordinate eigenspace transformations of the images, because these were made available at the SISAP site. The database was divided into one section with 762 objects, and one with 254 objects. Of the 762 objects, we used 64 as pivots and indexed the remaining 698. We evaluated the indexes using the other 254 objects as queries, and measured the wall clock time and number of distance cal-

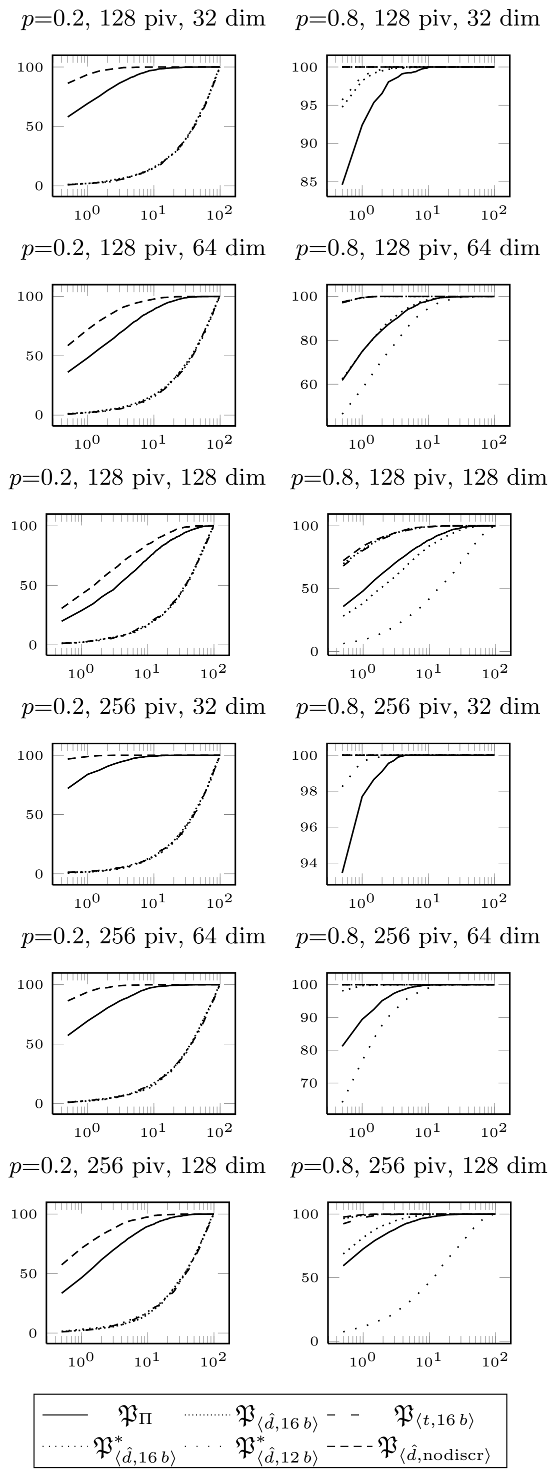

Figure 3: Index performance on 2-NN queries on uniformly generated L_p spaces with fractional p. On y axis: avg. % of true query result found. On x axis: % of max distance calculations.

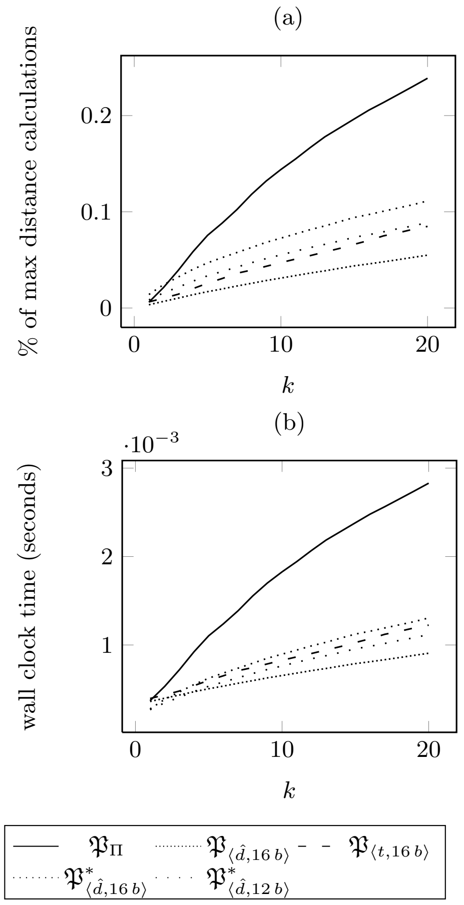

Figure 4: Index performance on L_2 space with 761-coordinate eigenspace transformations of face images. The plots show the amount of computation (y-axis) required to perfectly answer k-NN queries with varying values of k (x-axis).

culations necessary to perfectly answer k-NN queries with different values of k.

Documents (TREC). We also tested the methods on angle distance between vector representations of news documents from the Wall Street Journal, from the TREC data set [7]. There were 25 276 documents in the SISAP data. Of these, we used 128 as pivots, 1003 as queries, and indexed the remaining 24 145. The top part of Fig. 5 shows performance of range queries with a radius set to capture exactly 0.035 % of the database. The bottom part shows results for 5-NN queries.

Normalized edit distance (non-metric). Fig. 6 shows results of range queries (radius 1) on a space with normalized edit distance [10] between 40 000 words selected randomly from the Spanish dictionary file in the SISAP data. We used 32 and 128 pivots, and measured average percentage of the true query result found vs. percentage of the indexed objects having the true distance to the query calculated.

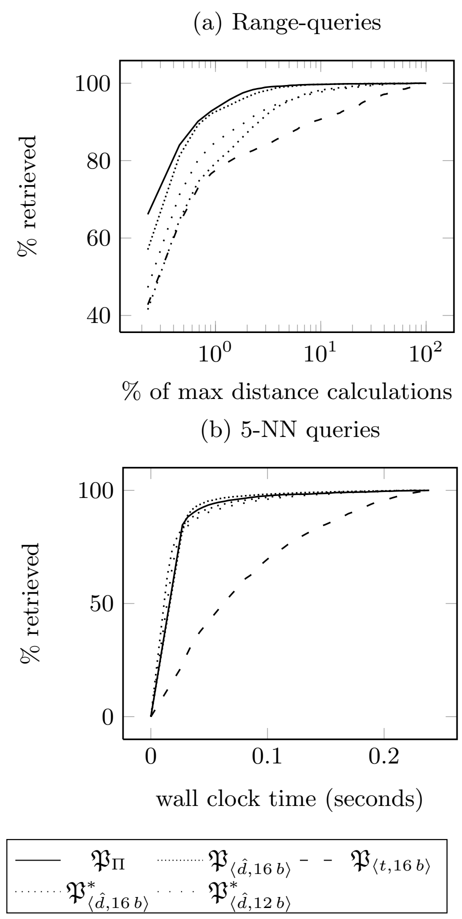

Figure 5: Index performance on angle distance between vector representations of news documents.

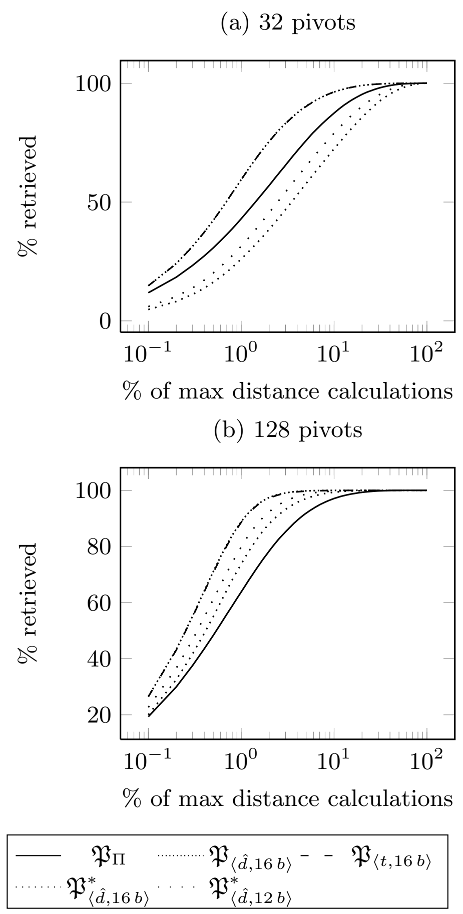

Figure 6: Index performance on range queries (radius 1) on normalized edit distance between 40 000 words from a Spanish dictionary.

5.3 Summary of results

The query performance of $\mathfrak{P}_{\langle \hat{d},16\,\mathrm{b}\rangle}$, the unfair (memory-wise) version of $\mathfrak{P}_{\hat{d}}$, was better than that of the baseline $\mathfrak{P}_{\Pi}$ in the following experiments: unit cube L_2 (Fig. 1), unit cube $L_{0.8}$ (Fig. 3, right column), face images L_2 (Fig. 4) and normalized edit distance (Fig. 6). It was roughly equal to the baseline on clustered L_2 (Fig. 2) and in one experiment with document angle distance (Fig. 5, bottom). It was slightly worse than the baseline in the other experiment with document angle distance (Fig. 5, top). On unit cube $L_{0.2}$ (Fig. 3, left column) $\mathfrak{P}_{\langle \hat{d},16\,\mathrm{b}\rangle}$, $\mathfrak{P}^*_{\langle \hat{d},16\,\mathrm{b}\rangle}$, $\mathfrak{P}^*_{\langle \hat{d},12\,\mathrm{b}\rangle}$ and $\mathfrak{P}_{\langle t,16\,\mathrm{b}\rangle}$ all performed much worse than the baseline, and in fact behaved much like random schedules. To investigate, we looked at the coefficients of the models (before discretization) and found that the models invariably had a very large positive or negative value for one coefficient, with the rest of the coefficients being close to 0. This caused our discretization scheme to break down. We included the undiscretized, unfair (memory-wise) $\mathfrak{P}_{\langle \hat{d},\mathrm{nodiscr}\rangle}$ in Fig. 3, and saw that it outperformed all other methods, including the baseline $\mathfrak{P}_{\Pi}$.

$\mathfrak{P}^*_{\langle \hat{d},16\,\mathrm{b}\rangle}$ and $\mathfrak{P}^*_{\langle \hat{d},12\,\mathrm{b}\rangle}$, the two fair (memory-wise) versions of $\mathfrak{P}_{\hat{d}}$, performed equal to or worse than $\mathfrak{P}_{\langle \hat{d},16\,\mathrm{b}\rangle}$ in all cases. Their performance was better than the baseline on unit cube L_2 (Fig. 1), face images L_2 (Fig. 4) and normalized edit distance with 128 pivots (Fig. 6, bottom). On unit cube $L_{0.8}$ (Fig. 3, right column), their performance was sometimes better and sometimes worse than the baseline. Their performance was roughly equal to the baseline on clustered L_2 with 128 pivots (Fig. 2, bottom), and in one experiment with document angle distance (Fig. 5, bottom). They fared worse than baseline on clustered L_2 with 32 pivots, in the other experiment on document angle distance (Fig. 5, top) and on normalized edit distance with 32 pivots (Fig. 6, top). The poor performance of $\mathfrak{P}^*_{\langle \hat{d},16\,\mathrm{b}\rangle}$ and $\mathfrak{P}^*_{\langle \hat{d},12\,\mathrm{b}\rangle}$ in 32-pivot experiments is likely due to the fact that very few pivots will then remain after memory-equalization.

The probability-based, unfair (memory-wise) $\mathfrak{P}_{\langle t,16\,\mathrm{b}\rangle}$ showed roughly the same performance as $\mathfrak{P}_{\langle \hat{d},16\,\mathrm{b}\rangle}$ on unit cube L_2 (Fig. 1), on clustered L_2 (Fig. 2), on unit cube $L_{0.2}$ and $L_{0.8}$ (Fig. 3), and on normalized edit distance (Fig. 6). It fared worse than $\mathfrak{P}_{\langle \hat{d},16\,\mathrm{b}\rangle}$ on face images L_2 (Fig. 4), where it beat the baseline, and on document angle distance (Fig. 5), where it was beaten by the baseline.

In the one experiment where we timed the index building, $\mathfrak{P}_{\langle \hat{d},16\,\mathrm{b}\rangle}$ took about 38 times as long build its index as $\mathfrak{P}_{\Pi}$.

6. DISCUSSION

The regression-based indexing scheme has some downsides and limitations. First, the index is very expensive to build, as we have described in section 4.4. This may make our approach infeasible for some applications. If the index building can be amortized over a large number of queries, or scheduled to periods of downtime, high index building cost may be acceptable.

A second limitation is that the use of regression brings with it some dependencies on the properties of the distance space to be indexed. This may make regression-based indexes difficult to apply out-of-the-box in practice. Regression models will generally give more accurate predictions and inferences when the data they are fitted and tested on satisfies the underlying assumptions of the models (as described in section 4). It is not trivial to measure the extent to which the assumptions are satisfied, but some standard diagnostic plots are recommended [6]. Having tentatively looked at plots of residuals vs. fitted values and residuals vs. normal distribution quantiles, it appears to us that the document angle distance data (Fig. 5), where $\mathfrak{P}_{\langle \hat{d}, 16\,\mathrm{b}\rangle}$ fared tolerably and $\mathfrak{P}_{\langle t, 16\,\mathrm{b}\rangle}$ fared poorly, exhibit more deviation than the face image L_2 data (Fig. 4), where they both fared well compared to the baseline. This may indicate that $\mathfrak{P}_t$ is more vulnerable to deviation from the model assumptions than $\mathfrak{P}_{\hat{d}}$. A possible explanation for the higher vulnerability of $\mathfrak{P}_t$ is that $\mathfrak{P}_t$ depends on the additional assumption that the errors are normally distributed.

The importance of the satisfaction of model assumptions also means that we should be especially careful when drawing general conclusions from the performance of regression-based indexing on synthetic data sets.

7. CONCLUSION

We have introduced a regression-based indexing scheme that in several of our experiments, also in some with real-world data, provided better query performance than the permutation based scheme [3] we used as a baseline, in some cases even with our scheme's pivot set reduced in order to equalize memory consumption. This indicates that $\mathfrak{P}_{\hat{d}}$ can be a competitive indexing scheme in terms of query performance, and calls for a more thorough investigation.

Disadvantages of our scheme include costly computation involved in building the index and vulnerability to deviation from the model assumptions in the distance spaces to be indexed. The latter consideration highlights the need to evaluate indexing schemes, and especially regression-based ones, on diverse selections of real-world data sets.

8. ACKNOWLEDGEMENTS

The authors would like to thank Svein Erik Bratsberg and Jon Marius Venstad for assistance and discussions.

9. REFERENCES

[1] V. Athitsos, J. Alon, S. Sclaroff, and G. Kollios. BoostMap: An embedding method for efficient nearest neighbor retrieval. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30(1):89–104, 2008.

[2] E. Chávez and G. Navarro. A probabilistic spell for the curse of dimensionality. In *Proc. 3rd Workshop on Algorithm Engineering and Experimentation (ALENEX)*, 2001.

[3] E. Chávez, K. Figueroa, and G. Navarro. Effective proximity retrieval by ordering permutations. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30(9):1647–1658, 2008.

[4] E. Chávez and G. Navarro. Probabilistic proximity search: Fighting the curse of dimensionality in metric spaces. *Information Processing Letters*, 85(1):39–46, 2003.

[5] E. Chávez, G. Navarro, R. A. Baeza-Yates, and J. L. Marroquín. Searching in metric spaces. *ACM Computing Surveys*, 33(3):273–321, 2001.

[6] J. J. Faraway. *Linear Models with R.* Chapman & Hall/CRC Press, 2005.

[7] D. Harman. Overview of the third text retrieval conference. In *Proc. Third Text REtrieval Conf. (TREC-3)*, 1995.

[8] M. L. Hetland. The basic principles of metric indexing. In *Swarm Intelligence for Multi-objective Problems in Data Mining.* Springer-Verlag, 2009.

[9] G. R. Hjaltason and H. Samet. Index-driven similarity search in metric spaces. *ACM Transactions on Database Systems*, 28(4):517–580, 2003.

[10] A. Marzal and E. Vidal. Computation of normalized edit distance and applications. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(9):926–932, 1993.

[11] M. L. Micó, J. Oncina, and E. Vidal. A new version of the nearest-neighbour approximating and eliminating search algorithm (AESA) with linear preprocessing time and memory requirements. *Pattern Recognition Letters*, 15(1):9–17, 1994.

[12] R. Paredes and G. Navarro. Optimal incremental sorting. In *Proc. 8th Workshop on Algorithm Engineering and Experiments (ALENEX)*, 2006.

[13] M. Patella and P. Ciaccia. Approximate similarity search: A multi-faceted problem. *Journal of Discrete Algorithms*, 7(1):36–48, 2009.

[14] P. Phillips, H. Wechsler, J. Huang, and P. Rauss. The FERET database and evaluation procedure for face recognition algorithms. *Image and Vision Computing Journal*, 16(5):295–306, 1998.

[15] R Development Core Team. *R: A Language and Environment for Statistical Computing.* R Foundation for Statistical Computing, Vienna, Austria, 2010. ISBN 3-900051-07-0.

[16] T. Skopal and B. Bustos. On nonmetric similarity search problems in complex domains. *ACM Computing Surveys*, to appear.