# Approximate Furthest Neighbor in High Dimensions

Rasmus Pagh, Francesco Silvestri, Johan Sivertsen, and Matthew Skala

IT University of Copenhagen

**Abstract.** Much recent work has been devoted to approximate nearest neighbor queries. Motivated by applications in recommender systems, we consider *approximate furthest neighbor* (AFN) queries. We present a simple, fast, and highly practical data structure for answering AFN queries in high-dimensional Euclidean space. We build on the technique of Indyk (SODA 2003), storing random projections to provide sublinear query time for AFN. However, we introduce a different query algorithm, improving on Indyk's approximation factor and reducing the running time by a logarithmic factor. We also present a variation based on a query-independent ordering of the database points; while this does not have the provable approximation factor of the query-dependent data structure, it offers significant improvement in time and space complexity. We give a theoretical analysis, and experimental results.

## 1 Introduction

Similarity search is concerned with locating elements from a set $S$ that are close to a given query $q$. The query $q$ can be thought of as criteria we would like returned items to satisfy approximately. For example, if a customer has expressed interest in a product $q$, we may want to recommend other, similar products. However, we do not want to recommend products that are *too* similar, since that would not significantly increase the probability of a sale. Among the points that satisfy a near neighbor condition ("similar"), we would like to return those that also satisfy a furthest-point condition ("not too similar"), without explicitly computing the set of all near neighbours and then searching it.

In this paper we focus on the problem of returning a furthest point, referred to as "furthest neighbor" by analogy with nearest neighbor. In particular, we consider point sets in $d$-dimensional Euclidean space ($\ell_2^d$). We argue that the exact version of this problem would also solve exact similarity search in $d$-dimensional Hamming space, and thus is as difficult as that problem. The reduction follows from the fact that the complement of every sphere in Hamming space is also a sphere. That limits the hope we may have for an efficient solution to the exact version, so we consider the *$c$-approximate furthest neighbor* ($c$-AFN) problem where the task is to return a point $x'$ with $d(q, x') \geq \max_{x \in S} d(q, x)/c$, with $d(x, u)$ denoting the distance between two points. We will pursue randomized solutions having a small probability of not returning a $c$-AFN. The success probability can be made arbitrarily close to 1 by repetition.

Another use for AFN comes from data structures that solve the approximate near neighbor ($(c, r)$-ANN) problem via locality-sensitive hashing. A known issue with the locality-sensitive hash approach to ANN is that if a point happens to be very close to the query, then nearly every hash function will return it. These duplicates cost significant time to filter out. AFN could offer a way to reduce such duplication. If each hash function preferentially returns the points furthest away from the query among the points it would otherwise return, then perhaps we could reduce the cost of filtering out duplicate results while still hitting all desired points with good probability.

We describe the data structure and query algorithm in section 2.2. The data structure itself is closely similar to one proposed by Indyk [10], but our query algorithm differs. It returns the $c$-approximate furthest neighbor, for any $c > 1$, with probability at least 0.72. When the number of dimensions is $O(\log n)$, our result requires $\tilde{O}(n^{1/c^2})$ time per query and $\tilde{O}(n^{2/c^2})$ total space, where $n$ denotes the input size. The $\tilde{O}()$ notation omits polylog terms. Theorem 2 gives bounds in the general case.

Our data structure requires more than linear space when $c < \sqrt{2}$, and there are theoretical reasons why $\sqrt{2}$ may be an important boundary for all data structures that solve this problem. In section 2.2 we give a preliminary result showing that a data structure for $c$-AFN must store at least $\min\{n, 2^{\Omega(d)}\} - 1$ data points when $c < \sqrt{2}$.

In section 3 we provide experimental support of our data structure by testing it, and some modified versions, on real and randomly-generated data sets. In practice, we can achieve approximation factors significantly below the $\sqrt{2}$ theoretical result, even with a simplified version of the algorithm that saves time and space by examining candidate points in a query-independent order. We can also achieve good approximation in practice with significantly fewer projections and points examined than the worst-case bounds suggested by the theory. Our techniques are much simpler to implement than existing methods for $\sqrt{2}$-AFN, which generally require convex programming [6, 15]. Our techniques can also be extended to general metric spaces.

## 1.1 Related work

*Exact furthest neighbor.* In two dimensions the furthest neighbor problem can be solved in linear space and logarithmic query time using point location in a furthest point Voronoi diagram (see e.g. de Berg et al. [3]). However, the space usage of Voronoi diagrams grows exponentially with the number of dimensions, making this approach impractical in high dimensions. Indeed, an efficient data structure for the *exact* furthest neighbor problem in high dimension would lead to surprising algorithms for satisfiability [19], so barring a breakthrough in satisfiability algorithms we must assume that such data structures are not feasible.

Further evidence of the difficulty of exact furthest neighbor is the following reduction from Goel, Indyk, and Varadarajan [9]: Given a set $S \subseteq \{0, 1\}^d$ and a query vector $q \in \{0, 1\}^d$, a furthest neighbor (in Euclidean space) from $-q$ is a vector in $S$ of minimum Hamming distance to $q$. That is, exact furthest neighbor is at least as hard as exact nearest neighbor in $d$-dimensional Hamming space, generally believed to be hard for large $d$ and worst-case data.

*Approximate furthest neighbor.* Bespamyatnikh gives a dynamic data structure for the *c-approximate* furthest neighbor problem; however, its query time is exponential in the dimension [4]. Indyk [10] avoids this exponential dependence. More precisely, Indyk showed how to solve a *fixed radius* version of the problem where given a parameter $r$ the task is to return a point at distance at least $r/c$ given that there exist one or more points at distance at least $r$. He then gives a solution to the furthest neighbor problem with approximation factor $c+\delta$, where $\delta > 0$, by reducing it to queries on many copies of that data structure. The overall result is space $O(dn^{1+1/c^2} \log^{(1-1/c)/2}(n) \log_{1+\delta}(d) \log\log_{1+\delta}(d))$ and query time $O(dn^{1/c^2} \log^{(1-1/c)/2}(n) \log_{1+\delta}(d) \log\log_{1+\delta}(d))$. While our new data structure uses the same basic method as Indyk, multiple random projections to one dimension, we are able to avoid the fixed radius version entirely and get a single and simpler data structure that works for all radii. Moreover, being interested in static queries, we are able to reduce the space to $\tilde{O}(dn^{2/c^2})$.

*Methods based on an enclosing ball.* Goel et al. [9] show that a $\sqrt{2}$-approximate furthest neighbor can always be found on the surface of the minimum enclosing ball of $S$. More specifically, there is a set $S^*$ of at most $d+1$ points from $S$ whose minimum enclosing ball contains all of $S$, and returning the furthest point in $S^*$ always gives a $\sqrt{2}$-approximation to the furthest neighbor in $S$. This method is *query independent* in the sense that it examines the same set of points for every query. Conversely, Goel et al. [9] show that for a random data set consisting of $n$ (almost) orthonormal vectors, finding a $c$-approximate furthest neighbor for a constant $c < \sqrt{2}$ gives the ability to find an $O(1)$-approximate near neighbor. Since it is not known how to do that in time $n^{o(1)}$ it is reasonable to aim for query times of the form $n^{f(c)}$ for approximation $c < \sqrt{2}$.

*Applications in recommender systems.* Several papers on recommender systems have investigated the use of furthest neighbor search [16, 17]. However, these works are not primarily concerned with (provable) efficiency of the search. Other related works in recommender systems include those of Abbar et al. [1] and Indyk et al. [11], which use core-set techniques to return a small set of recommendations no two of which are too close. In turn, core-set techniques also underpin works on approximating the minimum enclosing ball [2, 13].

## 2 Algorithms and analysis

### 2.1 Provably good furthest neighbor data structure

Our data structure works by choosing a random line and storing the order of the data points along it. Two points far apart on the line are at least as far apart in the original space. So given a query we can find the points furthest from the query on the projection line, and take those as candidates to be the furthest point in the original space. We build several such data structures and query them in parallel, merging the results.

Given a set $S \subseteq \mathbb{R}^d$ of size $n$ (the input data), let $\ell = 2n^{1/c^2}$ (the number of random lines) and $m = 1 + e^2 \ell \log^{c^2/2-1/3} n$ (the number of candidates to be examined at query time), where $c > 1$ is the desired approximation factor. We pick $\ell$ random vectors $a_1, \ldots, a_\ell \in \mathbb{R}^d$ with each entry of $a_i$ coming from the standard normal distribution $N(0,1)$. We use $\arg\max^m_{x \in S} f(x)$ for the set of $m$ elements from $S$ that have the largest values of $f(x)$, breaking ties arbitrarily.

For any $1 \leq i \leq \ell$, we let $S_i = \arg\max^m_{x \in S} a_i \cdot x$ and store the elements of $S_i$ in sorted order according to the value $a_i \cdot x$. Our data structure for $c$-AFN consists of $\ell$ subsets $S_1, \ldots, S_\ell \subseteq S$, each of size $m$. Since these subsets come from independent random projections, they will not necessarily be disjoint in general; but in high dimensions, they are unlikely to overlap very much. This data structure is essentially that of Indyk [10]; our technique differs in the query procedure, given by Algorithm 1.

**Algorithm 1** Query-dependent approximate furthest neighbor

```
 1: initialize a priority queue of (point, integer) pairs, indexed by real keys
 2: for $i = 1$ to $\ell$ do
 3:     compute and store $a_i \cdot q$
 4:     create an iterator into $S_i$, moving in decreasing order of $a_i \cdot x$
 5:     get the first element $x$ from $S_i$ and advance the iterator
 6:     insert $(x, i)$ in the priority queue with key $a_i \cdot x - a_i \cdot q$
 7: end for
 8: $rval \leftarrow \perp$
 9: for $j = 1$ to $m$ do
10:     extract highest-key element $(x, i)$ from the priority queue
11:     if $rval = \perp$ or $x$ is further than $rval$ from $q$ then
12:         $rval \leftarrow x$
13:     end if
14:     get the next element $x'$ from $S_i$ and advance the iterator
15:     insert $(x', i)$ in the priority queue with key $a_i \cdot x' - a_i \cdot q$
16: end for
17: return $rval$
```

Our algorithm succeeds if and only if $S_q$ contains a $c$-approximate furthest neighbor. We now prove that this happens with constant probability.

We make use of the following standard lemmas that can be found, for example, in the work of Datar et al. [7] and Karger, Motwani, and Suden [12], respectively.

**Lemma 1 (See Section 3.2 of Datar et al. [7]).** *For every choice of vectors $x, y \in \mathbb{R}^d$:*

$$\frac{a_i \cdot (x - y)}{||x - y||_2} \sim N(0,1). \qquad (1)$$

**Lemma 2 (see Lemma 7.1.3 in Karger, Motwani, and Suden [12]).** *For every $t > 0$, if $X \sim N(0,1)$ then*

$$\frac{1}{\sqrt{2\pi}} \cdot \left(\frac{1}{t} - \frac{1}{t^3}\right) \cdot e^{-t^2/2} \leq \Pr[X \geq t] \leq \frac{1}{\sqrt{2\pi}} \cdot \frac{1}{t} \cdot e^{-t^2/2} \tag{2}$$

The next lemma follows, as suggested by Indyk [10, Claims 2-3].

**Lemma 3.** *Let $p$ be a furthest neighbor from the query $q$ with $r = ||p-q||_2$, and let $p'$ be a point such that $||p'-q||_2 < r/c$. Let $\Delta = rt/c$ with $t$ satisfying the equation $e^{t^2/2}t^{c^2} = n/(2\pi)^{c^2/2}$ (i.e., $t = O\left(\sqrt{\log n}\right)$). Then, for a sufficiently large $n$, we get*

$$\Pr_a\left[a \cdot (p'-q) \geq \Delta\right] \leq \frac{\log^{c^2/2-1/3} n}{n} \tag{3}$$

$$\Pr_a\left[a \cdot (p-q) \geq \Delta\right] \geq (1-o(1))\frac{1}{n^{1/c^2}}\,. \tag{4}$$

*Proof.* Let $X \sim N(0,1)$. By Lemma 1 and the right part of Lemma 2, we get for a point $p'$ that

$$\begin{aligned}\Pr_a\left[a \cdot (p'-q) \geq \Delta\right] &= \Pr_a\left[X \geq \Delta/||p'-q||_2\right] \leq \Pr_a\left[X \geq \Delta c/r\right] \\ &\leq \frac{1}{\sqrt{2\pi}}\frac{e^{-t^2/2}}{t} \leq \left(t\sqrt{2\pi}\right)^{c^2-1}\frac{1}{n} \leq \frac{\log^{c^2/2-1/3} n}{n}.\end{aligned}$$

The last step follows because $e^{t^2/2}t^{c^2} = n/(2\pi)^{c^2/2}$ implies that $t = O\left(\sqrt{\log n}\right)$, and holds for a sufficiently large $n$. Similarly, by Lemma 1 and the left part of Lemma 2, we have for a furthest neighbor $p$ that

$$\begin{aligned}\Pr_a\left[a \cdot (p-q) \geq \Delta\right] &= \Pr_a\left[X \geq \Delta/||p-q||_2\right] = \Pr_a\left[X \geq \Delta/r\right] \\ &\geq \frac{1}{\sqrt{2\pi}}\left(\frac{c}{t} - \left(\frac{c}{t}\right)^3\right)e^{-t^2/(2c^2)} \geq (1-o(1))\frac{1}{n^{1/c^2}}.\end{aligned}$$

□

**Theorem 1.** *The data structure when queried by Algorithm 1 returns a c-AFN of a given query with probability $1 - 2/e^2 > 0.72$ in $O(n^{1/c^2}(d + \log^{c^2/2+2/3} n))$ time per query. The data structure requires $O(n^{1+1/c^2}(d + \log n))$ preprocessing time and total space*

$$O\left(\min\left\{dn^{2/c^2}\log^{c^2-1/3} n, dn + n^{2/c^2}\log^{c^2-1/3} n\right\}\right)\,. \tag{5}$$

*Proof.* The space required by the data structure is the space required for storing the $\ell$ sets $S_i$. If for each set $S_i$ we store the $m \leq n$ points and the projection values, then $O(\ell m d)$ memory words are required. On the other hand, if pointers to the input points are stored, then the total required space is $O(\ell m + nd)$.

Both representations are equivalent, and the best one depends on the value of $n$ and $d$. The claim on the space requirements follows. The preproceesing time is dominated by the computation of the $n\ell$ projection values and by the sorting for computing the sets $S_i$. Finally, the query time is dominated by the at most $2m$ insertion or deletion operations on the priority queue and by the search in $S_q$ for the furthest neighbor.

We now upper bound the success probability. As in the statement of Lemma 3, we let $p$ denote a furthest neighbor from $q$, $r = ||p - q||_2$, $p'$ be a point such that $||p' - q||_2 < r/c$, and $\Delta = rt/c$ with $t$ such that $e^{t^2/2}t^{c^2} = n/(2\pi)^{c^2/2}$. The query succeeds if: (i) $a_i(p-q) \geq \Delta$ for at least one projection vector $a_i$, and (ii) the (multi)set $S_n = \{p' | \exists i : a_i(p' - q) \geq \Delta, ||p' - q||_2 < r/c\}$ contains at most $m - 1$ points (i.e., there are at most $m - 1$ near points whose distances from the query is at least $\Delta$ in some projections). If (i) and (ii) hold, then the set $S_Q$ must contain the furthest neighbor $p$ since there are at most $m - 1$ points near to $q$ with projection values larger than the maximum projection value of $p$. Note that we do not consider points at distance larger than $r/c$ but smaller than $r$: they are $c$-approximate furthest neighbors of $q$ and can only increase the success probability of our data structure.

By Lemma 3, event (i) happens with probability $1/n^{1/c^2}$. Since there are $\ell = 2n^{1/c^2}$ independent projections, this event does not happen with probability at most $(1 - 1/n^{1/c^2})^{2n^{1/c^2}} \leq 1/e^2$. For a point $p'$ at distance at most $r/c$ from $q$, the probability that $a_i(p' - q) \geq \Delta$ is less than $(\log^{c^2/2-1/3} n)/n$ for Lemma 3. Since there are $\ell$ projections of $n$ points, the expected number of such points is $\ell \log^{c^2/2-1/3} n$. Then, we have that $S$ has size larger than $m - 1$ with probability at most $1/e^2$ by the Markov inequality. Note that a Chernoff bound cannot be used since there exists a dependency among the projections under the same random vector $a_i$. By a union bound, we can therefore conclude that the algorithm succeeds with probability at least $1 - 2/e^2 \geq 0.72$. □

## 2.2 A lower bound on the approximation factor

In this section, we show that a data structure aiming at an approximation factor less than $\sqrt{2}$ must use space $\min\{n, 2^{\Omega(d)}\} - 1$ on worst-case data. The lower bound holds for those data structures that compute the approximate furthest neighbor by storing a suitable subset of the input points.

**Theorem 2.** *Consider any data structure $\mathcal{D}$ that computes the c-AFN of an n-point input set $S \subseteq \mathbb{R}^d$ by storing a subest of the data set. If $c = \sqrt{2}(1 - \epsilon)$ with $\epsilon \in (0, 1)$, then the algorithm must store at least $\min\{n, 2^{\Omega\left(\epsilon^2 d\right)}\} - 1$ points.*

*Proof.* We prove by contradiction that any data structure requiring less than $\min\{n, 2^{\Omega\left(\epsilon^2 d\right)}\} - 1$ input points cannot return a $\sqrt{2}(1-\epsilon)$-approximation. Suppose there exists a set $S'$ of size $r = 2^{\Omega\left(\epsilon'^2 d\right)}$ such that for any $x \in S'$ we have $(1 - \epsilon') \leq ||x||_2^2 \leq (1 + \epsilon')$ and $x \cdot y \leq 2\epsilon'$, with $\epsilon' \in (0, 1)$. We will later prove that such a set exists.

Assume $n \leq r$. Consider the input set $S$ consisting of $n$ arbitrary points of $S'$ and set the query $q$ to $-x$, where $x$ is an input point not in the data structure. The furthest neighbor is $x$ and it is at distance at least $||x-(-x)||_2 \geq 2\sqrt{1-\epsilon'}$. On the other hand, for $y \in S\backslash\{x\}$, we get

$$||y-(-x)||_2^2 = ||x||_2^2 + ||y||_2^2 + 2x \cdot y \leq 2(1+\epsilon') + 4\epsilon'.$$

Therefore, the point returned is at least a $c'$ approximation with

$$c' \leq \sqrt{2}\sqrt{\frac{1-\epsilon'}{1+3\epsilon'}}. \tag{6}$$

The claim follows by setting $\epsilon' = \sqrt{(2\epsilon-\epsilon^2)/(1+3(1-\epsilon)^2)}$.

Assume now that $n > r$. Without loss of generality, let $n$ be a multiple of $r$. Consider as input set the set $S$ containing $n/r$ copies of each vector in $S'$, each copy expanded by a factor $i$ for any $i \in [1, n/r]$; specifically, let $S = \{ix | \forall x \in S', \forall i \in [1, n/r]\}$. Let the query $q$ be $-hx$, where $x$ is a point not in the data structure and $h$ is the largest integer such that $hy$, with $y \in S'$, is in the data structure. The furthest neighbor in $S$ is at distance at least $2h\sqrt{1-\epsilon'}$. On the other hand, every point in the data structure is at distance at most $h\sqrt{2(1+\epsilon')+4\epsilon'^2}$. We then get the same approximation factor $c'$ given in equation 6, and the claim follows by suitably setting $\epsilon'$.

The existence of the set $S'$ of size $r$ follows from the Johnson-Lindenstrauss lemma [14]. Specifically, consider an orthornormal base $x_1, \ldots x_r$ of $\mathbb{R}^r$. Since $n = \Omega\left(\log r/\epsilon^2\right)$, by the Johnson-Lindenstrauss lemma there exists a linear map $f(\cdot)$ such that $(1-\epsilon')||x_i - x_j||_2^2 \leq ||f(x_i) - f(x_j)||_2^2 \leq (1+\epsilon)||x_i - x_j||_2^2$ and $(1-\epsilon') \leq ||f(x_i)||_2^2 \leq (1+\epsilon')$ for any $i, j$. We also have that $f(x_i) \cdot f(x_j) = (||f(x_i)||_2^2 + ||f(x_j)||_2^2 - ||f(x_i) - f(x_j)||_2^2)/2$, and hence $-2\epsilon \leq f(x_i) \cdot f(x_j) \leq 2\epsilon$. It then suffices to set $S'$ to $\{f(x_1), \ldots, f(x_r)\}$. □

The upper bound on space translates into a lower bound for the query time in data structures for AFN which are query independent. Indeed, the lower bound translates into the number of points that must be read by each query. However, this does not apply for query dependent data structures.

## 3 Experiments

To test the algorithm and confirm both its correctness and practicality we implemented several variations in both the C and F# programming languages. This code is available on request. Our C implementation is structured as an alternate index type for the SISAP C library [8], returning the furthest neighbor instead of the nearest.

We selected four databases for experimentation: the "nasa" and "colors" vector databases from the SISAP library, and two randomly generated databases of $10^5$ 10-dimensional vectors each, one using a multidimensional normal distribution and one uniform on the unit cube. The 10-dimensional random distributions were intended to represent realistic data, but their intrinsic dimensionality is significantly higher than what we would expect to see in real-life applications.

For each database and each choice of $\ell$ from 1 to 30 and $m$ from 1 to $4\ell$, we made 1000 approximate furthest neighbor queries. To provide a representative sample over the randomization of both the projection vectors and the queries, we used 100 different seeds for generation of the projection vectors, and did 10 queries (each uniformly selected from the database points) with each seed. We computed the approximation achieved, compared to the true furthest neighbor found by brute force, for every query. The resulting distributions for the uniform, normal, and nasa databases are summarized in Figures 1–3.

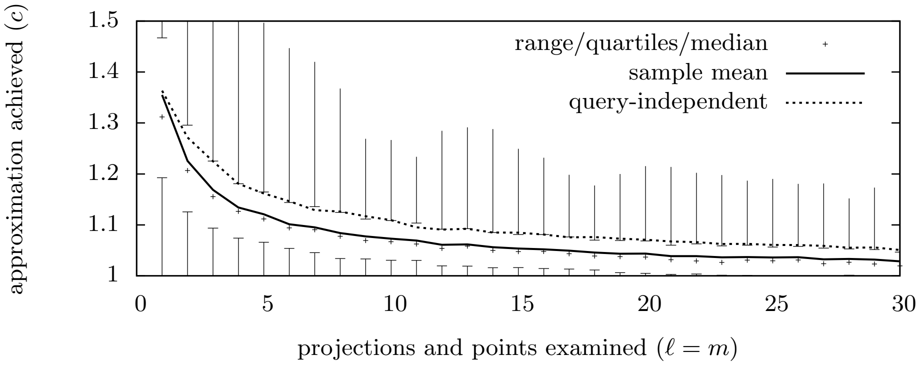

**Fig. 1.** Experimental results for 10-dimensional uniform distribution

We omit a similar figure from our experiment on the colors database because the result was of little interest: it apparently contains a few very extreme outliers, making the furthest neighbor problem too easy to meaningfully test the algorithm. We also ran some informal experiments on higher-dimensional random vector databases (with 30 and 100 dimensions, in particular) and saw approximation factors very close to those achieved for 10 dimensions.

*$\ell$ vs. $m$ tradeoff.* The two parameters $\ell$ and $m$ both improve the approximation as they increase, and they each have a cost in the time and space bounds. The best tradeoff is not clear from the analysis. We chose $\ell = m$ as a typical value, but we also collected data on many other parameter choices.

Figure 4 offers some insight into the tradeoff: since the cost of doing a query is roughly proportional to both $\ell$ and $m$, we chose a fixed value for their product, $\ell \cdot m = 48$, and plotted the approximation results in relation to $m$ given that. As the figure shows, the approximation factor does not change much with the tradeoff between $\ell$ and $m$.

*Query-independent ordering.* The furthest-neighbor algorithm described in our theoretical analysis examines candidates for the furthest neighbor in a *query*

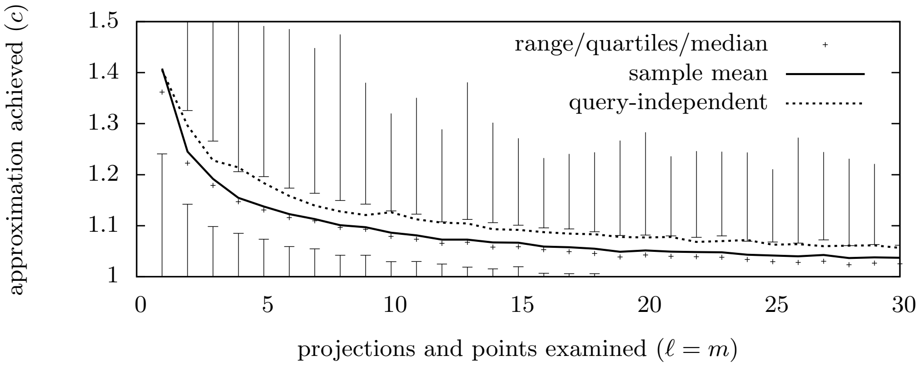

**Fig. 2.** Experimental results for 10-dimensional normal distribution

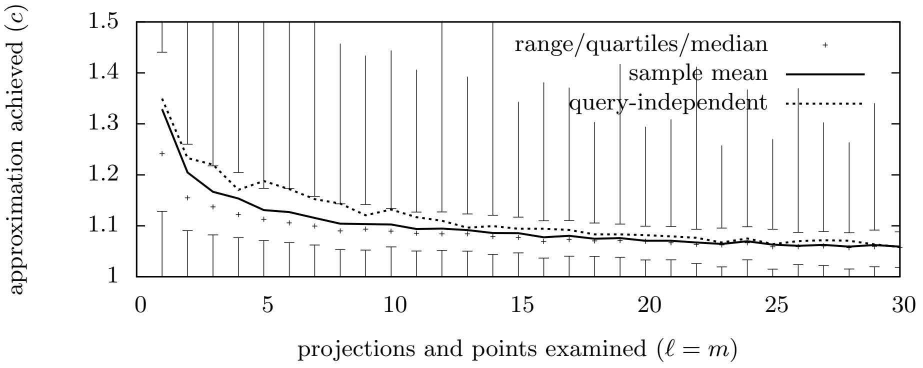

**Fig. 3.** Experimental results for SISAP nasa database

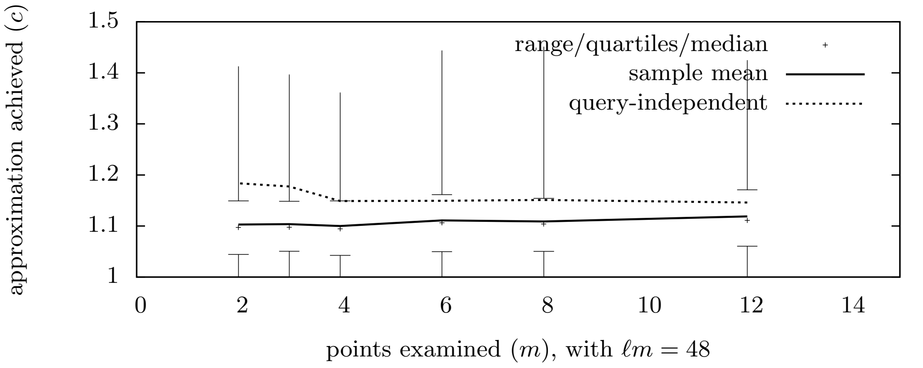

**Fig. 4.** The tradeoff between $\ell$ and $m$

*dependent* order. It seems intuitively reasonable that the search will usually examine points in a very similar order regardless of the query: first those that are outliers, on or near the convex hull of the database, and then working its way inward. Maybe there could be a single generic ordering of the points that would serve reasonably well for all queries?

We implemented a modified version of the algorithm in which the index stores a single ordering of the points. Given a set $S \subseteq \mathbb{R}^d$ of size $n$, for each point $x \in S$ let $\mathit{key}(x) = \max_{i \in 1 \ldots \ell} a_i \cdot x$. The key for each point is its greatest projection value on any of the $\ell$ randomly-selected projections. The data structure stores points (all of them, or enough to accomodate the largest $m$ we plan to use) in order of decreasing key value: $x_1$, $x_2$, … where $\mathit{key}(x_1) \geq \mathit{key}(x_2) \geq \cdots$.

The query simply examines the first $m$ points in this *query independent* ordering and returns the one furthest from the query point. Sample mean approximation factor for this algorithm in our experiments is shown by the dotted lines in Figures 1–4.

*Variations on the algorithm.* We have experimented informally with a number of practical improvements to the algorithm. The most significant is to use the rank-based *depth* of projections rather than the projection value. In this variation we sort the points by their projection value for each $a_i$. The first and last point then have depth 0, the second and second-to-last have depth 1, and so on up to the middle at depth $n/2$. We find the minimum depth of each point over all projections and store the points in a query independent order using the minimum depth as the key. This approach generally yields slightly better approximations, but is more complicated to analyze. A further improvement is to break ties in the minimum depth by count of how many times that depth is achieved, giving more priority to investigating points that repeatedly project to extreme values.

The number of points examined $m$ can be chosen per query and even during a query, allowing for interactive search. After returning the best result for some $m$, the algorithm can continue to a larger $m$ for a possibly better approximation factor on the same query. The smooth tradeoff we observed between $\ell$ and $m$ suggests that choosing an $\ell$ during preprocessing will not much constrain the eventual choice of $m$.

*Discussion.* The main experimental result is that the algorithm works very well for the tested datasets in terms of returning good approximations of the furthest neighbor. Even for small $\ell$ and $m$ the algorithm returns good approximations. Another result is that the query independent algorithm returns points only slighly worse than the query dependent. The query independent algorithm is simpler to implement, it can be queried in time $O(m)$ as opposed to $O(m \log \ell + m)$ and uses only $O(m)$ storage. In many cases these advances more than make up for the slightly worse approximation observed in these experiments. However, by Theorem 2, to guarantee $\sqrt{2} - \epsilon$ approximation the query-independent ordering version would need to store and read $m = n - 1$ points.

In data sets of high intrinsic dimensionality the furthest point from a query may not be much further than any randomly selected point, and we can ask whether our results are any better than a trivial random selection from the database. The intrinsic dimensionality statistic $\rho$ of Chávez and Navarro [5] provides some insight into this question. Skala gives a formula for its value on a multidimensional normal distribution [18, Theorem 2.10], which yields $\rho = 9.768\ldots$ for the 10-dimensional distribution used in Figure 2. With the definition $\rho = \mu^2/2\sigma^2$, this means the standard deviation of a randomly selected distance will be about 32% of the mean distance. Our experimental results come much closer than that to the true furthest distance, and so are non-trivial.

The concentration of distances in data sets of high intrinsic dimensionality reduces the usefulness of approximate furthest neighbor. Thus, although we observed similar values of $c$ in higher dimensions to our 10-dimensional random vector results, random vectors of higher dimension may represent a case where $c$-approximate furthest neighbor is not a particularly interesting problem. Fortunately, vectors in a space with many dimensions but low intrinsic dimensionality, such as the colors database, are more representative of real application data, and our algorithms performed well on such data sets.

## 4 Conclusions and future work

We have proposed a data structure for AFN with theoretical and experimental guarantees. Although we have proved that it is not possible to use less than $\min\{n, 2^{\Omega(d)}\} - 1$ total space when the $c$ approximation factor is less than $\sqrt{2}$, it is an open problem to close the gap between this lower bound and the space requirements of our result. Another interesting problem is to apply our data structure to improve the output sensitivity of near neighbor search based on locality-sensitive hashing. We conjecture that, by replacing each hash bucket with an AFN data structure with suitable approximation factors, it is possible to control the number of times each point in $S$ is reported.

Our data structure extends naturally to general metric spaces. Instead of computing projections with dot product, which requires a vector space, we could choose some random pivots and order the points by distance to each pivot. The query operation would be essentially unchanged. Analysis and testing of this extension is a subject for future work.

## Acknowledgement

The research leading to these results has received funding from the European Research Council under the European Union's Seventh Framework Programme (FP7/2007-2013) / ERC grant agreement no. 614331.

## References

1. Abbar, S., Amer-Yahia, S., Indyk, P., Mahabadi, S.: Real-time recommendation of diverse related articles. In: Proc. 22nd International Conference on World Wide Web (WWW). pp. 1–12 (2013)

2. Bădoiu, M., Clarkson, K.L.: Optimal core-sets for balls. Computational Geometry 40(1), 14 – 22 (2008)
3. de Berg, ., Cheong, O., van Kreveld, M., Overmars, M.: Computational Geometry: Algorithms and Applications. Springer-Verlag TELOS, 3rd ed. edn. (2008)
4. Bespamyatnikh, S.N.: Dynamic algorithms for approximate neighbor searching. In: Proceedings of the 8th Canadian Conference on Computational Geometry (CCCG-96). pp. 252–257. Carleton University (Aug 12–15 1996)
5. Chávez, E., Navarro, G.: Measuring the dimensionality of general metric spaces. Tech. Rep. TR/DCC-00-1, Department of Computer Science, University of Chile (2000)
6. Clarkson, K.L.: Las Vegas algorithms for linear and integer programming when the dimension is small. Journal of the ACM (JACM) 42(2), 488–499 (1995)
7. Datar, M., Immorlica, N., Indyk, P., Mirrokni, V.S.: Locality-sensitive hashing scheme based on p-stable distributions. In: Proc. 20 Annual Symposium on Computational Geometry (SoCG). pp. 253–262 (2004)
8. Figueroa, K., Navarro, G., Chávez, E.: Metric spaces library (2007), online `http://www.sisap.org/Metric_Space_Library.html`.
9. Goel, A., Indyk, P., Varadarajan, K.: Reductions among high dimensional proximity problems. In: Proc. 12th ACM-SIAM Symposium on Discrete Algorithms (SODA). pp. 769–778 (2001)
10. Indyk, P.: Better algorithms for high-dimensional proximity problems via asymmetric embeddings. In: Proc. 14th ACM-SIAM Symposium on Discrete Algorithms (SODA). pp. 539–545 (2003)
11. Indyk, P., Mahabadi, S., Mahdian, M., Mirrokni, V.S.: Composable core-sets for diversity and coverage maximization. In: Proc. 33rd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS). pp. 100–108. ACM (2014)
12. Karger, D., Motwani, R., Sudan, M.: Approximate graph coloring by semidefinite programming. Journal of the ACM (JACM) 45(2), 246–265 (1998)
13. Kumar, P., Mitchell, J.S., Yildirim, E.A.: Approximate minimum enclosing balls in high dimensions using core-sets. Journal of Experimental Algorithmics 8, 1–1 (2003)
14. Matoušek, J.: On variants of the Johnson-Lindenstrauss lemma. Random Structures and Algorithms 33(2), 142–156 (2008)
15. Matoušek, J., Sharir, M., Welzl, E.: A subexponential bound for linear programming. Algorithmica 16(4-5), 498–516 (1996)
16. Said, A., Fields, B., Jain, B.J., Albayrak, S.: User-centric evaluation of a k-furthest neighbor collaborative filtering recommender algorithm. In: Proc. Conference on Computer Supported Cooperative Work (CSCW). pp. 1399–1408 (2013)
17. Said, A., Kille, B., Jain, B.J., Albayrak, S.: Increasing diversity through furthest neighbor-based recommendation. Proceedings of the WSDM Workshop on Diversity in Document Retrieval (DDR) 12 (2012)
18. Skala, M.A.: Aspects of Metric Spaces in Computation. Ph.D. thesis, University of Waterloo (2008)
19. Williams, R.: A new algorithm for optimal constraint satisfaction and its implications. In: Proc. 31st International Colloquium on Automata, Languages and Programming (ICALP). pp. 1227–1237 (2004)