Approximating High-Dimensional Range Queries with kNN Indexing Techniques

Michael A. Schuh[1†], Tim Wylie[2], Chang Liu[1], and Rafal A. Angryk[3]

[1] Dept. of Computer Science, Montana State University, Bozeman, MT, 59717 USA.
[2] Dept. of Computer Science, University of Alberta, Edmonton, AB, Canada.
[3] Dept. of Computer Science, Georgia State University, Atlanta, GA, 30302 USA.
† `michael.schuh@cs.montana.edu`

Abstract. While k-nearest neighbor queries are becoming increasingly common due to mobile and geospatial applications, orthogonal range queries in high-dimensional data are extremely important in scientific and web-based applications. For efficient querying, data is typically stored in an index optimized for either kNN or range queries. This can be problematic when data is optimized for kNN retrieval and a user needs a range query or vice versa. There are naïve approaches, but their performance degrades quickly for large queries and high-dimensional spaces. Here, we address the issue of using a kNN-based index for range queries, as well as outline the general computational geometry problem of adapting these systems to range queries. We refer to these methods as space-based decompositions and provide a straightforward heuristic for this problem. Using iDistance as our applied kNN indexing technique, we also develop an optimal (data-based) algorithm designed specifically for its indexing scheme. We compare the performance of space-based and data-based methods to the suggested naïve approach in real world datasets and highlight the inherent difficulties of high-dimensional range queries. Results show our data-based algorithm consistently performs superior to the rest, and in worst case scenarios equals the naïve method.

1 Introduction

Modern society has grown dependent upon large-scale data mining applications. This reliance is only increasing as our ability to record and store vast quantities of rich data improves due to new technologies and new applications. Often this data is high-dimensional in nature, which can be challenging for basic use and understanding. For a user interacting with the data, there are two types of necessary queries: orthogonal range and k-nearest neighbor (kNN).

The use of nearest neighbor queries is essential in many applications such as geospatial, consumer, and mobile uses. These queries allow someone to search for the nearest store of interest to their location, or find a song, image, etc., of similar qualities to something they already know. Thus, many modern systems focus on this query methodology – especially within mobile and web-based applications. Orthogonal range queries differ in that they require a min and max range to be selected for every dimension of the query. This is used more often in scientific, user-defined, exploratory, and web search applications. Allowing the user to specify each dimensional range is important for capturing critical values or removing specificity entirely, *i.e.,* a wildcard search. This is also useful for data with non-numeric attributes where distance is meaningless.

Indexing methods allow large volumes of data to be stored in a way that is optimized to decrease retrieval time for their intended application. However, there may be times when another type of application (or query) is desired, but the data is stored in an inconvenient (and unoptimized) manner for this request. Here, we look at the situation that data is optimized for kNN retrieval, but range queries are also necessary. This assumes that we will not modify the current index, and our methods must use the inherent functionality of the existing indexing and retrieval mechanisms. We will focus on the state-of-the-art iDistance kNN index [1, 2], which has been used in a number of demanding applications, including large-scale image retrieval [3], video indexing [4], mobile computing [5], peer-to-peer systems [6], and video surveillance retrieval [7].

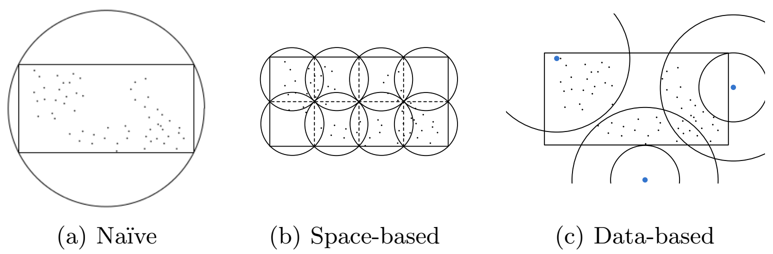

Fig. 1. Three different methods of retrieving range queries using iDistance. In (a) we naïvely encircle the range query. For (b) we cover the query space with overlapping smaller query spheres. In (c) we calculate the specific intersected iDistance partitions.

This work explores three methods for achieving range queries within a kNN indexing technique, shown as $2D$ examples in Figure 1. The first option is the naïve method, which has been suggested by many authors including the original iDistance publication [1], but this has drawbacks. In higher dimensions, only a few poorly chosen dimensional ranges can lead to the pessimal case as seen in Figure 3(d). Another possibility comes from the field of computational geometry. Since kNN queries are hyperspheres, we attempt to use a number of them to cover the hyperrectangle space of the range query. Unfortunately, this method breaks down in higher dimensions because it is an NP-hard problem to optimally cover the space. We provide a heuristic to accomplish this based on the ranges needed in each dimension. We assume that overlap between the query spheres is acceptable, and we are able to skirt the theoretical optimization problems based on minimum overlap, minimum number of queries, etc. These are rich and important areas of research [8], but are not critical to the purpose of this work and are therefore kept brief.

These first two methods can be classified as space-based decompositions because we seek to use one geometric object to cover the same space as another.

While these methods will work with any kNN system, the last method we develop is a novel data-based approach specific to iDistance. We refer to this as a data-based method because it uses the actual underlying data distributions rather than solely the coverage space of the range query. It also works directly on the iDistance index, and therefore other indexing schemes would require their own specific data-based range query algorithm.

Results highlight retrieval performance of range queries over three high-dimensional real-world datasets with existing kNN indexes. We find great variability in results over the various datasets which implies a certain degree of difficulty in performing useful range queries in high dimensional spaces, as well as a general lack of ability to anticipate results in practice. Regardless, we show the data-based method is the preferred choice for facilitating range queries as it out performs the naïve method in almost all situations.

This paper is organized as follows. Section 2 will briefly overview background and related work. Sections 3 and 4 will then cover the space-based methods, followed by the data-based method in Section 5. Then we present empirical evaluations and discussion in Section 6, and the paper concludes in Section 7.

2 Preliminaries

2.1 Indexing

For our study we will focus on iDistance, which is a modern kNN-based indexing method for high-dimensional data first proposed in 2001 [1]. It uses a filter-and-refine strategy based on a lossy transformation into an efficient one-dimensional B$^+$-tree for indexing and retrieval. iDistance is based on a Voronoi tessellation of the space [9], but for speed it approximates each of these areas with a hypersphere. These partitions $\mathfrak{P} = \{P_1, \ldots, P_M\}$ are stored based on their central reference point, $\mathfrak{O} = \{O_1, \ldots, O_M\}$. The number of partitions, M, can be arbitrarily set, as can the location of the reference points (and thereby resultant partitions). The data points are assigned to the closest partition by reference point distance. The radius of each partition is stored as $distmax$, which is the distance of the farthest point in that partition. Figure 2 shows an example of these elements as well as a query sphere and the areas of each partition to search.

Prior studies have shown that poor placement or an insufficient (or abundance) of partitions can greatly reduce the effectiveness of the algorithm [2, 10, 11]. The general best practice is to use a clustering algorithm, such as k-means which has similar spherical characteristics, to derive cluster centers to be used as reference points. It has also been shown to use a number of clusters directly related to the specific dataset size and dimensionality, with $2D$ partitions serving as a general rule of thumb [10].

For efficiency, the points are indexed in a one-dimensional B$^+$-tree [12], which allows iDistance to be built into modern databases. This lossy transformation separates partitions by a spacing constant c, which can be safely set to $\sqrt{D}$ since no two data points can be farther apart than this value in a normalized

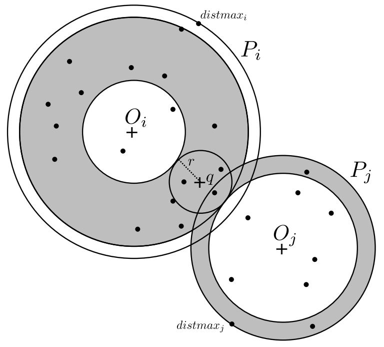

Fig. 2. A basic example of iDistance indexing partitions with a sample query q of radius r. Partitions P_i, P_j with reference points O_i, O_j, respectively.

dataspace. Then, for any point p assigned to a partition P_i, it is mapped in the B$^+$-tree to the value $y_p = c \cdot i + dist(O_i, p)$. This mapping can be seen in Figure 2, whereby concentric rings of data points share the same index value.

For a query, iDistance first finds all intersecting partitions and gets the points in the B$^+$-tree corresponding to this overlap in the filter step. Since several points may have the same value in the B$^+$-tree, iDistance must refine the results to ensure they are in the actual query sphere. If k results were not found, the radius of the query is increased and each partition is re-checked and possibly further searched. Only when the farthest item in the full set of k items is closer than the current query radius can the search be successfully terminated with the guaranteed exact results.

2.2 Geometric Queries

Formally, we will assume that any query is in a D dimensional space such that the set of dimensions is $\mathfrak{D} = \{d_1, \ldots, d_D\}$. A point kNN query is defined as a query point q with a radius r, which returns the k nearest elements within the ball $B(q, r)$ based on a specified distance metric $dist()$.

An orthogonal (or axis-aligned) range query in a D dimensional space is the set $\mathfrak{R} = \{R_{d_1}, \ldots R_{d_D}\}$. Each range R_{d_i} is a tuple of the minimum and maximum values that define the range over that dimension, *i.e.,* $R_{d_i} = \langle v_{min}, v_{max} \rangle$, so the range of dimension i is $[v_{min}, v_{max}]$. For convenience, we will also use the notation $\min(R_{d_i})$ for v_{min} in dimension i, and similarly for v_{max}. We will assume these definitions in the subsequent sections.

Orthogonal range queries, also known as window queries, are a well-studied problem that has spawned a lot of research for new data structures and algorithms for efficient retrieval [8]. Some of the more notable methods include kd-trees, multi-layered range trees, and R-trees [13]. Optimal retrieval time is provable with a tree construction, but requires greater storage than $O(N)$ [14]. Many of these methods achieve better time efficiency through additional space usage [8]. In large high-dimensional datasets, many of these methods can be too space inefficient to be practical. We focus on iDistance since it is a modern high-dimensional method, which is currently used widely in practice.

3 A Naïve Approach

The naïve method is easily adaptable for any kNN system. The simple idea is to encompass the desired range within a query sphere, retrieve all points inside this sphere, and then prune out the results not inside the original range. Figure 1(a) shows an example of this. What is immediately obvious is that the number of bad results returned grows quickly as the difference between the lengths of the hyperrectangle edges grows. This is especially problematic for wildcard-capable queries or several dimensions with wide ranges which can expand the query sphere to nearly the size of the entire dataspace, as shown in Figure 3(d).

The naïve method is a straight-forward approach that easily adapts a range query to a kNN system. The query point is the average of all the ranges, and the radius is from the center to a corner of the hyperrectangle. The value of each dimension d_i for the center c of the circumscribed sphere is calculated by the Equation 1, and the radius from the center to the corner by Equation 2.

$$c_i = \frac{\min(R_{d_i}) + \max(R_{d_i})}{2} \qquad (1) \qquad\qquad r^2 = \sum_{i=1}^{D} (\max(R_{d_i}) - c_i)^2 \qquad (2)$$

In the above equations, c_i is the coordinate value of the query center point in the i^{th} dimension ($0 \leq c_i \leq 1$ in a normalized space). After we have calculated the query sphere, the kNN algorithm works the same except we do not bound our return set to only the k closest items, and instead return everything found within the query sphere. An additional refinement step is then required to prune all points from the kNN query that are outside the original range query. Although this method is cited as an easy extension [1], its drawbacks make it impractical.

4 Space-Based Decomposition

Encircling a hyperrectangle can be inefficient with respect to the amount of area included outside the hyperrectangle. The optimal shape, resulting in minimal area included outside the hyperrectangle, is when the sides are equal and we have a hypercube. Our goal is to give a heuristic to decompose a range query into a set of smaller range queries that are closer to hypercubes, and thereby individually more efficient. Then each of these queries can be encircled and searched with any kNN system like the naïve method. An example is shown in Figure 1(b). This decomposition will result in less area searched overall, although the complexity will be higher due to the algorithm overhead and combination of multiple partially overlapping (and unbounded) kNN queries.

Sphere packing is a well-studied problem, but many problems related to packing and covering with overlapping spheres are still open such as the optimal 1-density above two dimensions [15]. The 1-density, δ_D^1 in D dimensions, is the volume covered by a single sphere in a space covered with unit spheres as shown in Figure 3(a). In $2D$, the optimal is $opt(\delta_2^1) = (3\sqrt(3) - \pi)/\pi \approx 0.6539$, and it is believed that in $3D$ the optimal is $opt(\delta_3^1) \approx 0.315$, which was determined by empirical methods [15]. It is also known that as D increases, the optimal 1-density decreases, and the limit is zero as the dimensionality goes to infinity. This becomes important in our application. We briefly outline the space-based decomposition problem in relation to range queries and give our heuristic which may be a polynomial-time approximation scheme (PTAS) while $hal_I(\delta_D^1) > 0$ for an instance I in D dimensions.

Definition 1. *Space-based Decomposition:*
Instance: *Given an orthogonal range query* $\mathfrak{R} = \{R_{d_1}, \dots R_{d_D}\}$ *in* D *dimensions where* $R_{d_i} = \langle v_{min}, v_{max} \rangle$ *is the minimum and maximum value allowed in the* i^{th} *dimension, and a* $K \in \mathbb{R}^+$.
Problem: *Find a set of spheres* $\mathfrak{S} = \{s_1, \dots, s_S\}$ *such that* $V_{\mathfrak{S}} = \cup_{i=1}^{|\mathfrak{S}|} Vol(s_i)$ *covers* $V_{\mathfrak{R}} = \cup_{i=1}^{|\mathfrak{R}|} Vol(R_{d_i})$*, the* $V_{\mathfrak{S}} - V_{\mathfrak{R}} \leq K$ *and for any* $s_i, s_j \in \mathfrak{S}$*, the 1-density* $hal_{s_i}(\delta_D^1) = hal_{s_i}(\delta_D^1) > 0$.

Essentially, we want to cover the range hyperrectangle with hyperspheres while guaranteeing that each sphere covers at least a volume of K which no other sphere covers, and that the total volume queried outside the hyperrectangle is less than J. Our method attempts to partition each dimension such that the resulting subrectangles are close to hypercubes. This guarantees that for each new sphere, we can bound the 1-density based on knowing the number of neighboring spheres and the amount of intersection. We note that all our spheres have the same radius, which is necessary in order to avoid a bin-packing problem. Thus, the spheres can be regarded as unit-spheres.

For a D-cube, we know that it has $2D$ sides, and thus $2D$ D-spheres which intersect the hypersphere covering that hyperrectangle. If we let each side be represented by x, then we know can calculate the 1-density for each sphere. The diameter of every sphere is $d = \sqrt{Dx^2}$. From this we can calculate the intersecting volume of the $2D$-spheres and subtract it from the volume of the hypercube $V_c = x^D$.

The overlap of a single sphere from one side is $\frac{1}{2D}(V_s - V_c)$ where V_s is the volume of a single sphere. Thus, the 1-density of our heuristic algorithm is $hal(\delta_2^1) = 2V_c - V_s + DO_s$ where O_s is the overlap volume of a sphere onto a neighboring sphere, which is zero in $2D$. The 1-density shrinks drastically in higher dimensions. For $2D$ the value is $2x^2 - \frac{x^2\pi}{2}$ and if $x = 1$ then $hal(\delta_2^1) = 2 - \pi/2 \approx .4292$ and is shown in Figure 3(a). Our heuristic is far less than the optimal $opt(\delta_2^1) \approx 0.6539$, which means we have more overlap in our query spheres. The optimal requires a difficult partitioning of the space in $2D$, and the optimal is unknown above $2D$.

Our heuristic tries to find the most economical way to split each dimension such that the resulting hyperrectangles are as close to hypercubes as possible. Based on an $\varepsilon > 0$, we can find the optimal number of divisions. We find the appropriate number of divisions, X_i, for a given dimensional range, R_i where $1 \leq i \leq D$, with Equation 3. We can see in Figure 3 how each extra power increases the number of sections created in a given dimension. For our current discussion, we ignore the real world effects of retrieving and combining multiple overlapping queries in an indexing system such as iDistance.

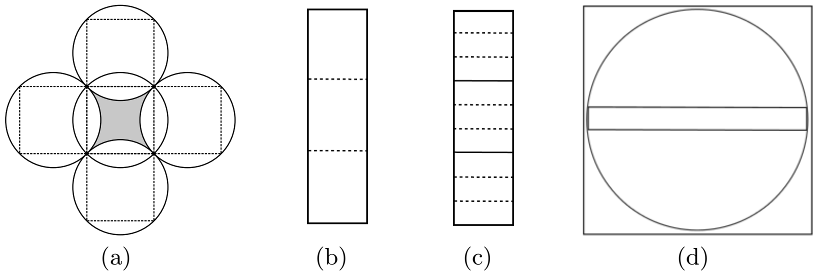

Fig. 3. (a) A $2D$ example of the 1-density in our heuristic. The shaded region is the 1-density of the circle around the center cube. (b) A range query with a base-3 split once into three sections and then (c) split once again creating nine total sections. (d) A normalized dataspace with a range query resulting in a poor naïve solution that queries most of the dataspace even though only a small area is necessary.

In order to find the best practical ratio with respect to ε, we chose to look at several logarithmic bases b. This means $b^{X_i^b}$ is the number of resulting sections. This also means we may find different values for X_i depending on which base we use, and we therefore add a superscript to denote the base. For our algorithm, we limit the bases to the first few primes to ensure that there are only a constant number of calculations each time while still satisfying ε. Given the splits, we want the ratio that is closest to one for each dimension. The ratio is given in Equation 4. Note that we can use a different ε for each dimension to achieve the best ratio nearest to one. For the complexity, simply using the smallest ε gives us the upper bound.

$$X_i^b = \left\lfloor log_b\left(\frac{R_i}{\varepsilon}\right)\right\rfloor, b = 2, 3, 5, 7, \ldots \quad (3) \qquad\qquad r_{ib} = \frac{R_i}{\varepsilon \cdot (b^{X_i^b})} \quad (4)$$

The ε term allows us to increase the number of spheres used and as we make them smaller we decrease the volume covered outside the range query. We can make this volume as small as desirable, and our running time is dependent on how many spheres there are. Thus, this may be a PTAS which is still exponential in ε. For a more practical heuristic we limit $b = \{2, 3, 5\}$ and set $\varepsilon = \min(\mathfrak{R})$. The value $\min(\mathfrak{R})$ is the smallest length of any side of the hyperrectangle, and thus a good basis for finding the closest dimensions to approximate a hypercube.

Although this method is straightforward and constant to calculate, the number of query spheres grows exponentially with respect to the number of dimensions split. This makes the general algorithm infeasible in any high-dimensional space when there is a lot of variation between the query ranges in each dimension. One could consider bounding the algorithm to only a few splits or dimensions, but this is non-trivial and changes the desired results. Further, we know that as the dimensionality increases, the overlap of the spheres also increases, which means it becomes less effective in higher dimensions because we are repeatedly re-searching parts of the dataspace. To be efficient, the indexing method would need to track already searched space. This would require a modification to the indexing system, which is impractical and beyond our scope.

5 Data-based Method

We have previously focused on methods for ensuring the same range query area in the dataspace is covered by queries. Instead, we now want to ignore the underlying space of the range query, and simply focus on the data that exists within this space. As you can see in Figure 1(c), given clustered data in the space, we only need to search where data exists. Since iDistance is an exact retrieval algorithm that indexes all data based on spherical partitions, we only need to identify which partitions the range query intersects, and to what extent they intersect.

Since our range queries are orthogonal, the closest point p in the axis aligned range $\mathfrak{R}$ to an outside reference point O_i is given on a per dimension (d) comparison. That is, we can calculate the closest point to O_i that is still in the range by looking at each dimension individually. We drop the i subscript here for clarity since we are using the location of the point for a given dimension as the subscript.

$$p_d(R_d, O_d) = \begin{cases} \min(R_d), & \text{if } O_d < \min(R_d) \\ \max(R_d), & \text{if } O_d > \max(R_d) \\ O_d, & \text{if } \min(R_d) \leq O_d \leq \max(R_d) \end{cases} \tag{5}$$

The closest point in $\mathfrak{R}$ is also the closest point in partition P_i that any data may be within the range. We can then take the distance $m = dist(O_i, p)$ and search the interval of $[distmax - m, distmax]$ in the B$^+$-tree – partition boundaries often further limit this interval. If m is greater than the $distmax$, then the range query does not intersect that partition. If it does, then we immediately know exactly how much and a single query will retrieve all the results.

There are a few special cases, such as if the partition completely encapsulates the range. In this case we can default to the naïve method to search (which would be equivalent here), but to be consistent with our algorithm we calculate the farthest point and the closest point within our range to the reference point. Equation 5 is the closest, but the farthest point can also easily be deduced in a similar manner. We then search the partition in the B$^+$-tree over the range [closest-point, farthest-point].

To enable this data-based range query feature, we had to modify essentially only one critical function from the original iDistance implementation. By modifying the *SearchO* function previously presented in [2], we use iDistance as a state-of-the-art kNN indexing algorithm to optimally facilitate and retrieve high-dimensional range queries. Presented in Algorithm 1, we now include this modification as the standard *SearchO* algorithm for range queries in our own open-source iDistance implementation[4]. Note this does not change any of the underlying indexing mechanisms.

[4] http://code.google.com/p/idistance/

Algorithm 1 Find all points within a specified range.

```
Input: $\mathfrak{R}$ is the set of ranges in each dimension.
1:  procedure RangeSearchO ($\mathfrak{R}$)
2:      for each $P_i \in \mathfrak{P}$ do                                  ▷ Filter Step
3:          closest-point ← ClosestPoint($\mathfrak{R}, O_i$)              ▷ Eq. 5
4:          $m \leftarrow dist(O_i$, closest-point)
5:          if $m \le distmax_i$ then
6:              farthest-point ← FarthestPoint($\mathfrak{R}, O_i$)
7:              if $distmax_i <$ farthest-point then
8:                  farthest-point ← $distmax_i$
9:              SearchInward($(c \cdot i)$ + farthest-point, $m$)
10:     for each $p \in S$ do                                               ▷ Refine Step
11:         if $p \notin \mathfrak{R}$ then
12:             $S \leftarrow S \backslash p$
```

6 Empirical Evaluation

We now test our three methods of kNN-based range query retrieval through general performance comparisons on real world data. Using best practices, we setup a standard iDistance index on each unit-space normalized dataset using k-meanss to derive $2D$ reference points. Results are presented for three commonly used public datasets of varying dimensionality and size. Synthetic datasets and other types (and amounts) of reference points were also researched here, but due to limited space and similar results, we exclude these from discussion.

To explicitly investigate the functionality of the methods, we performed range queries with a varying number of wide dimensions that cover the entire range $[0, 1]$. We randomly select 100 data points as queries to ensure that our query point distribution follows the underlying dataset distribution. For each experiment we show the number of candidate points returned during the filter step and the final result set size as a percentage of the dataset, as well as the total time taken (in milliseconds) to perform the query and return the final exact results. For space, we omit results for B$^+$-tree nodes accessed, as they directly mimic candidates accessed for all tests performed. This is expected behavior, but we have found it not always to be the case. We also omit results for the space-based decomposition method because while effective in low dimensions, or with only a few wide dimensions, it (as expected) quickly becomes inefficient and impractical in higher dimensional spaces.

Sequential scan (SS) is often used as a benchmark comparison for worst-case performance. It must check every data point, and even though it does not use the B$^+$-tree for retrieval, total tree nodes provides the appropriate worst-case comparison. Note that all data fits in main memory, so all experiments are compared without depending on the behaviors of specific hardware-based disk-caching routines. In real-life however, disk-based I/O bottlenecks are a common concern for inefficient retrieval methods. Therefore, in these experiments it is more important that the index properly filters out data points (fewer candidates) so that they do not have to be accessed whatsoever.

We begin with the UCI Corel Image Features dataset[5] (referred to herein as Color). This is the smallest of the three, containing 68,040 points in 32 dimensional space derived from HSV color histograms. Results are shown in Figure 4 for the Naïve and data-based (Data) methods compared to SS. We include the return set size (Set) in the candidates chart. Notice first that even the initial hypercube is retrieved much more efficiently by the data-based method, whereby the naïve method starts poorly and immediately degrades to searching almost the entire dataset. Once that happens, sequential scan is a better choice because it does not have the then unnecessary algorithmic overhead, which is evident by the total time taken for all methods. Also notice the lesser time required when the entire space is required to be searched upfront (all wide dims), rather than methodically determined to be searched anyways (e.g., 28 dims). As we said earlier however, filter capability is the most important factor here, and we see the data-based method gracefully degrades to the naïve (and SS) performance as the number of wide dimensions increases.

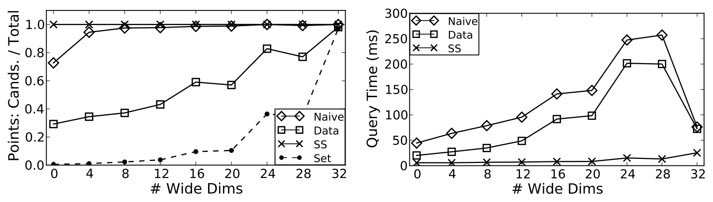

Fig. 4. Range queries with increasing number of wide dimensions for the Color dataset.

Next we tested the UCI YearPredictionMSD dataset[6] (Music), which has 90 dimensions and 515,345 data points representing musical characteristics of songs. Results are presented in Figure 5, and here we see the data-based method only performs slightly better than the naïve method. Notice the initial query already returns about 20% of the dataset, which could be affecting performance.

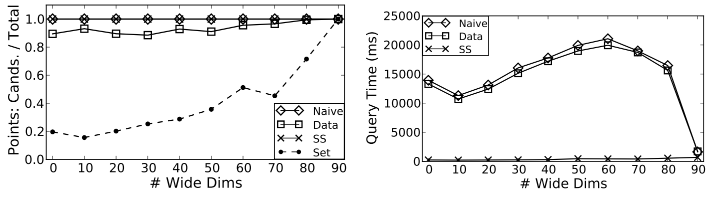

Fig. 5. Range queries with increasing number of wide dimensions for the Music dataset.

Lastly, Figure 6 presents the results for the "ANN_ SIFT1M" dataset [7] (Sift). The largest dataset we tested, it contains 1 million points and was specifically

[5] https://archive.ics.uci.edu/ml/datasets/Corel+Image+Features
[6] http://archive.ics.uci.edu/ml/datasets/YearPredictionMSD
[7] http://corpus-texmex.irisa.fr/

created for testing nearest neighbor retrieval on 128-dimensional SIFT feature vectors extracted from images. The results are quite different than the Music dataset, rejecting any purely "high-dimensional" performance factors in favor of more rich factors tied closely to the dataset characteristics. Here we see quite exceptional performance for the data-based method in both statistics. However, it is likely in part due to the extremely small result set size, which is on average only the data point the query is centered on. To our surprise, this holds true even with almost all wide dimensions in the query. This also explains why the time taken increases with all wide dimensions, rather than decrease like the other datasets, because it finally incurs a higher cost of a successful refinement step. By maintaining relatively high performance in retrieving a small set from a very large and high dimensional dataset, this suggests a promising use case for application of retrieving highly selective high-dimensional range queries.

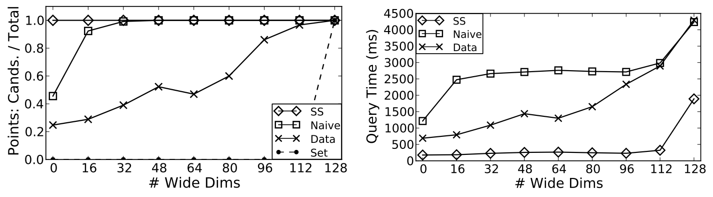

Fig. 6. Range queries with increasing number of wide dimensions for the Sift dataset.

We find varying results over all three datasets, but it is clear the data-based method is superior to the trivial naïve approach. We believe further investigation and analysis of these datasets could shed light on the results we find, and this will be true of any real world dataset. The Color dataset provides reasonable results we would expect to see, while the Music and Sift datasets seem to show opposite possible outcomes. We believe the Music dataset shows signs of dense areas given the poor filtering ability on all queries, while the Sift dataset appears extremely sparse and disconnected, since we are able to avoid most of the dataset while searching to return essentially nothing (one in a million).

7 Conclusions

This paper presents solutions for approximating orthogonal range (window) queries in kNN indexing systems. Using iDistance for the high-dimensional kNN indexing algorithm, we developed an optimal data-based range query algorithm that requires no modifications to the underlying index and retrieval mechanisms and no additional storage costs. Results show our novel data-based method greatly improves performance and efficiency of range queries in iDistance using the suggested naïve approach. While providing graceful degradation towards the naïve solution in worst-case scenarios, it in general promises exceptional filtering of the dataset even when many wide dimensions are specified, satisfying any practical wildcard-like searching. We also highlight the variability and difficulty of satisfying and testing high-dimensional range queries in real data.

While our method is optimal for iDistance, other indexing techniques will require their own data-based methods for supporting non-trivial range query retrieval. Beyond this, we are interested in future work surrounding the nature and usage of range queries in practice. For example, preliminary results in selectivity-based range queries yield drastically different results for different real world datasets, much like the return set sizes presented here. Other considerations include weighted range queries (hyper-ellipses), exploratory (or fuzzy boundary) range queries, and rank (distance-based) results, all of which could be performed from an existing kNN index.

References

1. Yu, C., Ooi, B.C., Tan, K.L., Jagadish, H.V.: Indexing the Distance: An Efficient Method to KNN Processing. In: Proc. of the 27th Int. Conf. on Very Large Data Bases. VLDB '01, San Francisco, CA, USA, Morgan Kaufmann Publishers Inc. (2001) 421–430
2. Jagadish, H.V., Ooi, B.C., Tan, K.L., Yu, C., Zhang, R.: iDistance: An adaptive B+-tree based indexing method for nearest neighbor search. ACM Trans. Database Syst. **30** (June 2005) 364–397
3. Zhang, J., Zhou, X., Wang, W., Shi, B., Pei, J.: Using high dimensional indexes to support relevance feedback based interactive images retrieval. In: Proc. of the 32nd inter. conf. on Very large data bases. VLDB '06 (2006) 1211–1214
4. Shen, H.T.: Towards effective indexing for very large video sequence database. In: SIGMOD Conference. (2005) 730–741
5. Ilarri, S., Mena, E., Illarramendi, A.: Location-dependent queries in mobile contexts: Distributed processing using mobile agents. IEEE TMC **5** (2006) 1029–1043
6. Doulkeridis, C., Vlachou, A., Kotidis, Y., Vazirgiannis, M.: Peer-to-peer similarity search in metric spaces. In: VLDB '07. (2007)
7. Qu, L., Chen, Y., Yang, X.: idistance based interactive visual surveillance retrieval algorithm. In: Intelligent Computation Technology and Automation. Volume 1 of ICICTA'08. (Oct 2008) 71–75
8. de Berg, M., Cheong, O., van Kreveld, M., Overmars, M.: Computational Geometry: Algorithms and Applications. 3rd edn. Springer (April 2008)
9. Aurenhammer, F.: Voronoi diagrams – a survey of a fundamental geometric data structure. ACM Comput. Surv. **23** (Sept 1991) 345–405
10. Schuh, M.A., Wylie, T., Banda, J.M., Angryk, R.A.: A comprehensive study of iDistance partitioning strategies for knn queries and high-dimensional data indexing. In: Proc. of the 29th British National Conf. on Databases. Volume 7968 of Lecture Notes in Computer Science., Springer-Verlag (2013) 238–252
11. Schuh, M.A., Wylie, T., Angryk, R.A.: Mitigating the curse of dimensionality for exact knn retrieval. In: Proceedings of the 1st International Workshop on High-Dimensional Data Mining. HDM'13 (2013) 1–12
12. Bayer, R., McCreight, E.: Organization and maintenance of large ordered indices. Acta Inform. **1** (1972) 173–189
13. Guttman, A.: R-trees: a dynamic index structure for spatial searching. In: Proc. of the ACM SIGMOD Inter. Conf. on Management of data. (1984) 47–57
14. Chazelle, B.: Lower bounds for orthogonal range searching: I. the reporting case. Journal of the ACM **37**(2) (Apr 1990) 200–212
15. Zhu, B.: On the 1-density of unit ball covering. CoRR **abs/0711.2092** (2007)