A Comprehensive Study of iDistance Partitioning Strategies for *k*NN Queries and High-Dimensional Data Indexing

Michael A. Schuh, Tim Wylie, Juan M. Banda, and Rafal A. Angryk

Montana State University, Bozeman, MT 59717-3880, USA
{michael.schuh, timothy.wylie, juan.banda, angryk}@cs.montana.edu

Abstract. Efficient database indexing and information retrieval tasks such as k-nearest neighbor (kNN) search still remain difficult challenges in large-scale and high-dimensional data. In this work, we perform the first comprehensive analysis of different partitioning strategies for the state-of-the-art high-dimensional indexing technique iDistance. This work greatly extends the discussion of why certain strategies work better than others over datasets of various distributions, dimensionality, and size. Through the use of novel partitioning strategies and extensive experimentation on real and synthetic datasets, our results establish an up-to-date iDistance benchmark for efficient kNN querying of large-scale and high-dimensional data and highlight the inherent difficulties associated with such tasks. We show that partitioning strategies can greatly affect the performance of iDistance and outline current best practices for using the indexing algorithm in modern application or comparative evaluation.

Keywords: iDistance, Large-scale, High-dimensional, Indexing, Retrieval, kNN

1 Introduction

Modern database-oriented applications are filled with rich information composed of an ever-increasing amount of large-scale and high-dimensional data. While storing this data is becoming more routine, efficiently indexing and retrieving it is still a practical concern. A frequent and costly retrieval task on these databases is k-nearest neighbor (kNN) search, which returns the k most similar records to any given query record. While all database management systems (DBMS) are highly optimized for a few dimensions, the traditional indexing algorithms (*e.g.*, the B-tree and R-tree families) degrade quickly as the number of dimensions increase, and eventually a sequential (linear) scan of every single record in the database becomes the fastest retrieval method.

Many algorithms have been proposed in the past with limited success for truly high-dimensional indexing, and this general problem is commonly referred to as *the curse of dimensionality* [4]. Practitioners often mitigate these issues through dimensionality reduction techniques (manual and automated) before using multi-dimensional indexing methods, or even adding application logic to combine multiple independent indexes or requiring user involvement during search. However,

modern applications are increasingly employing highly-dimensional techniques to effectively represent massive data, such as the highly popular 128-dimensional SIFT features [11] in Content-Based Image Retrieval (CBIR).

First published in 2001, iDistance [10, 20] specifically addressed kNN queries in high-dimensional space and has since proven to be one of the most efficient and state-of-the-art high-dimensional indexing techniques available for exact kNN search. In recent years, iDistance has been used in a number of demanding applications, including large-scale image retrieval [21], video indexing [15], mobile computing [8], peer-to-peer systems [6], and surveillance system video retrieval [14]. Unfortunately, no works to date have focused on developing methods of best practice for these modern applications.

This work methodically analyzes partitioning strategies with the goal of increasing overall performance efficiency of indexing and retrieval determined by the total tree nodes accessed, candidate records returned, and the time taken to perform a query. These metrics are used to quantitatively establish best practices and provide benchmarks for the comparison of new methods. We introduce a new and open-source implementation of the original iDistance algorithm[1] including detailed documentation, examples, visualizations, and extensive test scripts. We also contribute research-supporting code for pre-processing datasets and post-processing results, as well as all published algorithmic improvements.

The motivations addressed in the original iDistance publications have only increased in importance because of the ubiquity of rich high-dimensional and large-scale data for information retrieval, such as multimedia databases and the mobile computing market which have exploded in popularity since the last publication in 2005. While there is little doubt the algorithm remains effective and competitive, a more thorough investigation into performance-affecting criteria is needed to provide a basis for general capabilities and best practices. Without this study, it can be difficult to effectively use iDistance in application and reliably compare it to new methods in future research.

The rest of the paper is organized as follows. Section 2 highlights background and related works, an overview of iDistance is presented in Section 3, and experiments and results are presented in Section 4. We follow with a brief discussion of key findings and best practices in Section 5, and we close with our conclusions and future work in Section 6.

2 Background and Related Works

The ability to efficiently index and retrieve data has become a silent backbone of modern society, and it defines the capabilities and limitations of practical data usage. While the one-dimensional B^+-tree [2] is foundational to the modern relational DBMS, most real-life data has many dimensions (attributes) that would be better indexed together than individually. Mathematics has long-studied the partitioning of multi-dimensional metric spaces, most notably Voronoi Diagrams and the related Delaunay triangulations [1], but these theoretical solutions can be

[1] Publicly available at: http://code.google.com/p/idistance/

too complex for application. R-trees [7] were developed with minimum bounding rectangles (MBRs) to build a hierarchical tree of successively smaller MBRs containing objects in a multi-dimensional space, and R*-trees [3] enhanced search efficiency by minimizing MBR overlap. However, these trees (and most derivations) quickly degrade in performance as the dimensions increase [5, 13].

Recently, research has focused on creating indexing methods that define a one-way lossy mapping function from a multi-dimensional space to a one-dimensional space that can then be indexed efficiently in a standard B^+-tree. These lossy mappings require a filter-and-refine strategy to produce exact query results, where the one-dimensional index is used to quickly retrieve a subset of the data points as candidates (the filter step), and then each of these candidates is checked to be within the specified query region in the original multi-dimensional space (the refine step). Because checking the candidates in the actual dataspace is a costly task, the goal of the filter step is to return as few candidates as possible while retaining the exact results.

The Pyramid Technique [5] was one of the first prominent methods to effectively use this strategy by dividing up the d-dimensional space into $2d$ pyramids with the apexes meeting in the center of the dataspace. This was later extended by moving the apexes to better balance the data distribution equally across all pyramids [22]. For greater simplicity and flexibility, iMinMax(θ) [13, 16] was developed with a global partitioning line θ that can be moved based on the data distribution to create more balanced partitions leading to more efficient retrieval. The simpler transformation function also aids in faster filter-step calculations for finding candidate sets. Both the Pyramid Technique and iMinMax(θ) were designed for range queries in a multi-dimensional space, and extending to high-dimensional kNN queries is not a trivial task.

It should also be briefly noted that many other works are focused on returning approximate nearest neighbors [9, 18], but these are outside the scope of efficient exact kNN retrieval by iDistance presented in this paper.

3 iDistance

The basic concept of iDistance is to segment the dataspace into disjoint partitions, where all points in a specific partition are *indexed by their distance* ("iDistance") to the reference point of that partition. This results in a set of one-dimensional distance values, each related to one or more data points, for each partition that are all together indexed in a single standard B^+-tree. The algorithm was motivated by the ability to use arbitrary reference points to determine the (dis)similarity between any two data points in a metric space, allowing single dimensional ranking and indexing of data points no matter what the dimensionality of the original space [10]. The algorithm also contains several adjustable parameters and run-time options, making the overall complexity and performance highly dependent on the choices made by the user. Here we provide an overview of the algorithm and readers are encouraged to refer to the original works [10, 20] for details that are beyond the scope of our investigation.

3.1 Building the Index

The most important algorithmic option for iDistance is the partitioning strategy. The original works presented two types of abstract partitioning strategies: *space-based*, which assumes no knowledge of the actual data, and *data-based*, which adjusts the size and location of partitions based on the data distribution [10, 20]. For any strategy, every partition requires a representative reference point, and data points are assigned to the single closest partition in Euclidean distance.

A mapping scheme is required to create separation between the partitions in the underlying B^+-tree, ensuring any given index value represents a unique distance in exactly one partition. Given a partition P_i with reference point O_i, the index value y_p for a point p assigned to this partition is defined by Equation 1, where $dist()$ is any metric distance function, i is the partition index, and c is a constant multiplier for creating the partition separation. While constructing the index, each partition P_i records the distance of its farthest point as $distmax_i$.

$$y_p = i \times c + dist(O_i, p) \tag{1}$$

3.2 Querying the Index

The index should be built in such a way that the filter step returns the fewest possible candidate points without missing the true k-nearest neighbor points. Fewer candidates reduces the costly refinement step which must verify the true multi-dimensional distance of each candidate from the query point. Performing a query q with radius r consists of three basic steps: 1) determine the set of partitions to search, 2) calculate the search range for each partition in the set, and 3) retrieve the candidate points and refine by true distance.

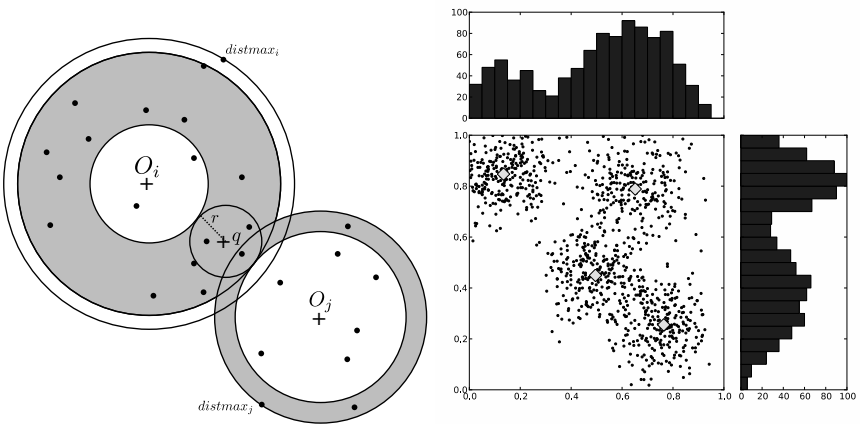

Fig. 1: (a) A query sphere q with radius r and the searched regions (shaded) in the two overlapping partitions P_i and P_j defined by their reference points O_i and O_j, and radii $distmax_i$ and $distmax_j$ respectively. (b) A scatter plot of a two dimensional dataset with four clusters, accompanied by each single dimensional histogram.

Figure 1(a) shows an example query point q with radius r contained completely within partition P_i and intersecting partition P_j, as well as the shaded ranges of each partition that need to be searched. For each partition P_i and its $distmax_i$, the query sphere (q, r) overlaps the partition if the distance from the edge of the query sphere to the reference point O_i is less than $distmax_i$, as defined in Equation 2. There are two possible cases of overlap: 1) q resides within P_i, or 2) q is outside of P_i, but the query sphere intersects it. In the first case, the partition needs to be searched both inward and outward from the query point over the range $(q \pm r)$, whereas an intersected partition is only searched inward from the edge of the partition to the farthest point of intersection. Equation 3 combines both overlap cases into a single search range for each partition.

$$dist(O_i, q) - r \leq distmax_i \tag{2}$$

$$[dist(O_i, q) - r, MIN(dist(O_i, q) + r, distmax_i)] \tag{3}$$

3.3 Partition Strategies

Space-based Strategies The only space-based methods presented in detail in previous works [10, 20] were *Center of Hyperplane* and *External Point*, which we refer to in this work as *Half-Points* (HP) and *Half-Points-Outside* (HPO), respectively. The HP method mimics the Pyramid-Technique [5] by placing reference points at the center of each dimensional edge of the data space with $2d$ partitions in d dimensions. The HPO method creates the same reference points, but then moves them outside of the dataspace by a preset distance to reduce the overlap volume between partitions. For example, in a 2D space such as Figure 1(b), HP would result in four partitions, based on reference points: $(0.0, 0.5)$, $(0.5, 0.0)$, $(1.0, 0.5)$, and $(0.5, 1.0)$, and HPO-10 (movement of 10.0) would result in reference points: $(-10.0, 0.5)$, $(0.5, -10.0)$, $(11.0, 0.5)$, and $(0.5, 11.0)$ respectively. Here we also introduce random reference point selection (RAND) to create any number of partitions located randomly in the dataspace. While this is a trivial strategy, it has not been shown before and greatly helps compare other strategies by providing a naïve benchmark.

Data-based Strategies The primary benefit of data-based methods is their adaptability to data distributions, which greatly increases retrieval performance in real-world settings. Two methods were originally introduced: *center of cluster* and *edge of cluster*, but only the *center of cluster* method was actually presented in published results [10, 20], which used algorithmically derived cluster centers as reference points to create cluster-based partitions in the dataspace.

Approximate cluster centers can be found through a variety of popular clustering algorithms, such as k-Means [12], BIRCH [23], *etc.*, and the original authors recommend (without explicit rationale) to use $2d$ as the number of partitions (clusters). They believed using the edges of clusters is intuitively more promising as it should reduce partition overlap (decreasing node accesses) and reduce the number of equi-distant points from any given reference point (decreasing candidates). Unfortunately, they leave us with only implementation

suggestions, such as "points on hyperplanes, data space corners, data points at one side of a cluster and away from other clusters, and so on" [10], but many of these methods are infeasible in high dimensions and were never presented.

4 Experiments and Results

We propose new iDistance partitioning strategies and methodically determine the effectiveness of various strategies over a wide range of dataset characteristics that lead to generalized conclusions about when and how to apply certain strategies (if at all). This not only depends on the dataset size and dimensionality, but also on additional knowledge possibly available, such as data distributions and clusters. We highlight these variabilities over extensive experiments that not only validate the results (and independent/unbiased reproducibility) of original research [10, 20], but also greatly extend the analyses through novel strategies and specially designed dataspaces.

Every run of our implementation of iDistance reports a set of statistics describing the index and query performance of that run. As an attempt to remove machine-dependent statistics, we use the number of B^+-tree nodes instead of page accesses when reporting query results and tree size. Tracking nodes accessed is much easier within the algorithm and across heterogeneous systems, and is still directly related to page accesses through the given machine's page size and B^+-tree leaf size. We primarily highlight three statistics from tested queries: 1) the number of candidate points returned during the filter step, 2) the number of nodes accessed in the B^+-tree, and 3) the time taken (in milliseconds) to perform the query and return the final results. Often we express the ratio of candidates and nodes over the total number of points in the dataset and the total number of nodes in the B^+-tree, respectively, as this eliminates skewed results due to varying the dataset.

The first experiments are on synthetic datasets (uniform and clustered) so we can properly simulate specific dataset conditions, and we later apply these results towards evaluating strategies on real world dataset. All artificial datasets are given a specified number of points and dimensions in the unit space $[0.0, 1.0]$. For clustered data we provide the number of clusters and the standard deviation of the independent Gaussian distributions centered on each cluster (in each dimension). For each dataset, we randomly select 500 points as kNN queries (with $k = 10$) for all experiments, which ensures that our query point distribution follows the dataset distribution.

Sequential scan 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, unless sequential scan runs significantly faster, there is a greater implied benefit when the index method does not have to access every data record, which could potentially be on disk.

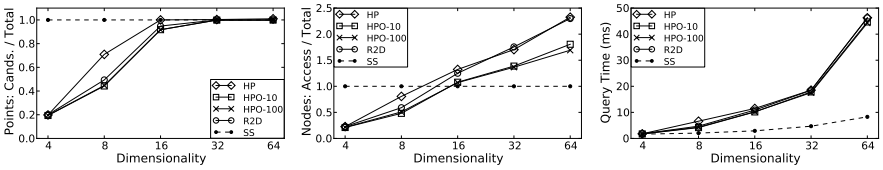

Fig. 2: Space-based methods on uniform data (10K) over dimensions.

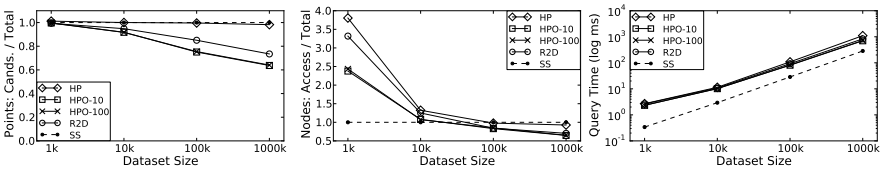

Fig. 3: Space-based methods on uniform data (16D) over dataset size.

4.1 Space-based Strategies in Uniform Data

Our first experiments compare *Sequential Scan* (SS) to space-based methods in uniform datasets ranging from 4 to 64 dimensions and 1,000 (1k) to 1 million (1000k) points. We present *Half-Points* (HP) and *Half-Points-Outside* (HPO), specifically HPO-10 and HPO-100, and also show the RAND method with an equivalent $2d$ reference points (R2D).

Figures 2 and 3 validate the original claim that HPO performs better than HP [10, 20], but surprisingly it also shows that R2D works better than HP. We can also see that a movement of 10.0 (HPO-10) outside of the dataspace is sufficient for performance improvements with HPO, and there is minimal gain thereafter. Although space-based methods take longer than SS in 16 dimensions (16D) or less, they access significantly less nodes and return fewer candidates. Note that it is possible to access the same nodes multiple times because data points from disjoint partitions can be stored in the same tree leaves. Another important performance factor is dataset size, shown in Figure 3 over a constant 16D. This can be linked to Figure 2 at 10k data points. We now log-transform the query time to show that as expected, larger datasets slow down all methods. However, sequential scan grows the fastest (with a linear increase), because at a certain point space-based strategies begin to properly filter the congested space and access less nodes while returning fewer candidates.

While still using uniform data, we investigate the effects of varying the number of reference points. Figures 4 and 5 look at the RAND method with 16 (R16), 64 (R64), 256 (R256), and 1024 (R1024) reference points. We also include dynamic methods of $2d$ over number of dimensions d (R2D) and $\sqrt{p}$ over number of points p (RP*), which are meant to better account for the specific dataspace characteristics. The results highlight the trade-off between dimensions of a space and total points, showing that as the number of dimensions increase, more partitions reduce the number of candidates, but also increase the nodes accessed and overall query time. Conversely, as the number of data points increase and dimensionality holds constant, kNN queries become more compact, and the number of candidates and nodes decreases leading to a shorter query time.

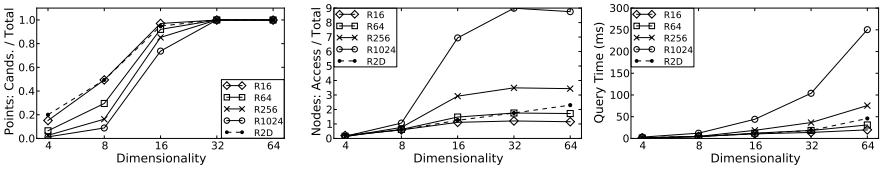

Fig. 4: Varying number of random ref. points on uniform data (10K) over dimensions.

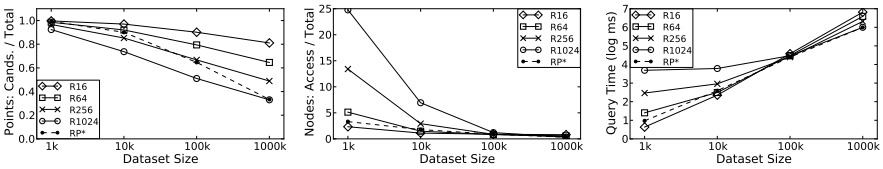

Fig. 5: Varying number of random ref. points on uniform data (10K) over dataset size.

4.2 The Transition to Clustered Data

Since most real-world data is not uniform, we turn our attention to clustered data and data-based partitioning strategies. As mentioned by the authors in the original iDistance publications [10, 20], data-adaptive indexing is the primary strength of iDistance, and we too show it greatly improves overall performance. We start by trying to better understand when data-based strategies overtake space-based strategies through varying cluster densities in the space, which has not been investigated previously. For each dataset, cluster centers (12 total) are randomly generated and then points are sampled with a standard deviation (*stdev*) ranging from 0.40 to 0.005 in each dimension of the 16D space with a total of 100k points equally distributed among clusters. We use the actual cluster centers – *True Centers* (TC) – as the only reference points. For comparison, we include *Half-Points* (HP) and *Sequential Scan* (SS) as baseline benchmarks. The RAND method was not included because it produces unpredictable results depending on the location of reference points and underlying data clusters.

In Figure 6, we can see the effect that cluster density has as the space transitions from very loose to extremely tight clusters. We do not report candidates because the results closely mirror the nodes accessed ratio. While using the true centers of the clusters as reference points quickly becomes the better technique, it eventually stalls out and fails to improve once the data is sufficiently dense – but notice that HP's performance steadily increases to near similar results. Since the space-based reference points are not bound to clusters, they continue to increase in effectiveness by searching smaller and smaller "slices" of each partition.

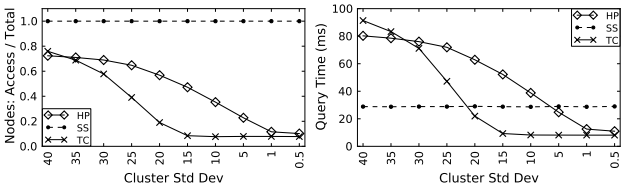

Fig. 6: Results over varying cluster density (by standard deviation).

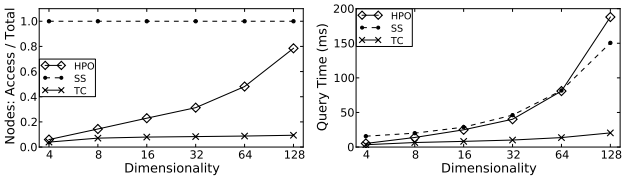

Fig. 7: Results over dimensions of 12 clusters with 0.1 standard deviation.

We can further see these trade-offs in Figure 7. Here we set the *stdev* of all 12 clusters to 0.1 and vary the dimensionality of 100k data points. The 12 equal-sized clusters seem to explain why TC stabilizes with around 8% (or 1/12) of the nodes accessed in both of these figures. In other words, the clusters become so dense that although the kNN queries rarely have to search outside of a single partition, they ultimately have to search through the entire partition containing the query. We confirm this in Figure 8, which shows the total partitions checked and candidates returned for three clustered datasets with 6 (TC6), 12 (TC12), and 24 (TC24) clusters over varying cluster density. Notice that all three start with accessing all partitions and most data points, but all converge to only one checked partition with the respective ratio of candidates.

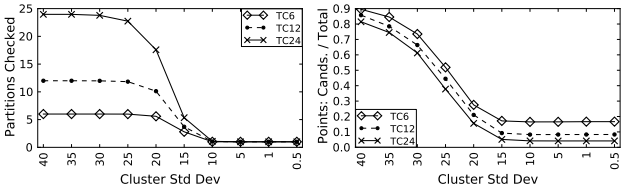

Fig. 8: Results of various numbers of clusters over cluster density.

4.3 Reference Points: Moving from Clusters

We now investigate more advanced data-based partitioning strategies using the *True Centers* (TC) of clusters as our benchmark reference points. Original works make mention of reducing partition overlap, and thereby increasing performance, by moving reference points away from each other [10, 20], but did not investigate it. This approach should perform better than TC because there will be less equi-distant points for each reference point, meaning the lossy transformation is less destructive for true data point similarity.

We present two strategies for moving reference points away from cluster centers. Since cluster centers are typically found by minimizing inter-cluster similarity while maximizing intra-cluster similarity, by moving reference points away from the cluster centers, one could hypothesize that there should be less equi-distant points in each partition and therefore a more discriminative one-dimensional B^+-tree index. The two methods are: 1) *Min-Edge* (M), moving towards the closest edge of the dataspace in any single dimension, and 2) *Random* (R), moving randomly in any/all dimensions. We specify movement by a total distance in the multi-dimensional dataspace and both methods are capable of pushing reference points outside of the dataspace – which makes the *Min-Edge*

method similar to *Half-Points Outside* (HPO). Using *Min-Edge* on the data in Figure 1(b) as an example, the upper-left cluster center will decrease along the x-axis, and the upper-right cluster will increase along the y-axis.

Figure 9 shows the ratio of candidates returned from the two cluster center movement methods (M and R), with movement distances of {0.025, 0.05, 0.1, 0.2, 0.4}, each compared to TC. Each method performs best with a movement distance of 0.2, as shown with TC in the third column chart for better readability. We can see that above 16D (with 12 clusters and 100k points) no methods seem to make a significant difference. However, lower dimensions do support our hypothesis that moving away from the centers can help. Figure 10 shows the same methods in 16D over a varying number of data points, and here we see the methods also become ineffective as the number of points increase.

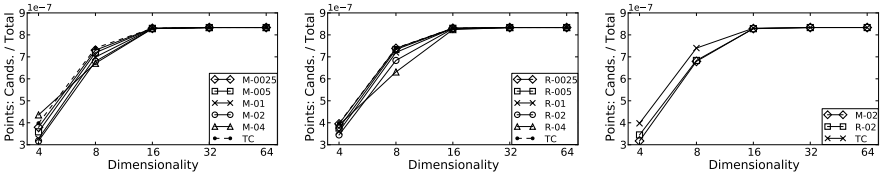

Fig. 9: Results of center movement methods.

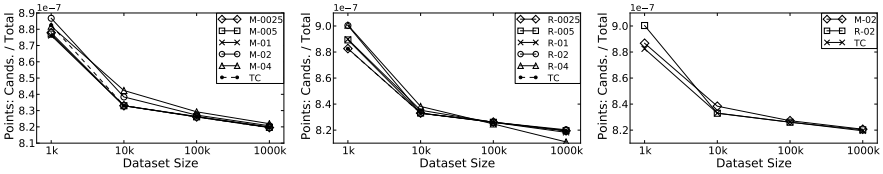

Fig. 10: Results of center movement methods.

4.4 Reference Points: Quantity vs. Quality

While we know that iDistance performs well on datasets with known clusters, a more common scenario is less knowledge of the data where the size and number of clusters are unknown. This is the focus of the original iDistance works, which suggest the use of any popular clustering algorithm as a pre-processor to identify more optimal reference point placements. The original publications used BIRCH [23] in 2001 [20] and k-Means [12] in 2005 [10].

In these experiments we investigate the effect of the number of provided clusters during our pre-processing with the k-Means algorithm. It should be stated that k-Means is known to be sensitive to the initial starting position of cluster centers, and does not ensure any balance between cluster populations. We use a standard MATLAB implementation and mitigate these inherent weaknesses by initializing our cluster centers on a randomly sampled data subset, and forcing all clusters to contain at least one point so the resultant reference points are not accidentally removed and ignored from the space. Although never discussed in previous works, we believe it is very important to address the case of non-empty clusters, especially when analyzing how well a certain number of reference points

perform. Otherwise, there is no guarantee that the specified number of reference points actually reflects the same number of partitions as intended.

The authors of iDistance originally suggested a general setting of $2d$ reference points – so k-Means with $k = 2d$ clusters – which also matches the space-based strategies [10, 20]. In Figure 11, we look at the performance of k-Means (KM) with d-relative clusters from $d/2$ to $4d$, in various dimensions over 100k points in 12 clusters. We also include *True Centers* (TC) as our current baseline benchmark, and k-Means with 12 clusters but without knowledge of the true cluster centers upon initialization (KM-12*). Notice the relatively equal nodes accessed ratio for all methods in higher dimensions, but the increase in overhead time taken for the methods with more clusters (partitions).

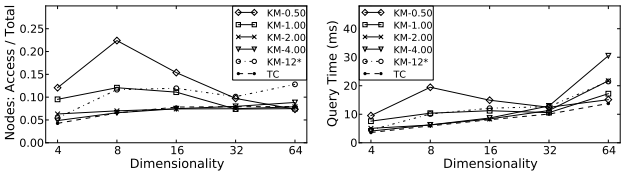

Fig. 11: Varying k-Means centers with 12 clusters (100K points) over dimensions.

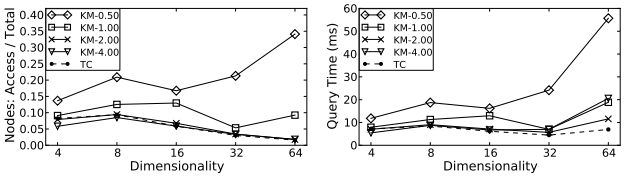

Fig. 12: Varying k-Means centers with d clusters (100K points) over dimensions.

An important realization here is how beneficial the knowledge of true cluster centers can be, as we see TC performs more consistently (and nearly always better) than other methods over all dimensions. The same can be seen in Figure 12, where we now generate d clusters in the dataset instead of only 12. However, here we see that in higher dimensions more clusters make a major difference for the number of nodes accessed, and $2d$ clusters seem in many cases to be an appropriate balance between the number of partitions and the time taken to search all the partitions, as both $1d$ and $4d$ clusters are equally slower. Also note that setting k to the number of known clusters for k-Means (KM-12* in Figure 11) does not guarantee performance because of the variability of discovered clusters from the k-Means algorithm.

Our final experiments use a real dataset to determine if any of our findings carry over from synthetic dataset studies. We use a popular real world dataset containing one million 128-dimensional SIFT feature vectors[2]. This dataset was recently used by the authors of the SIMP algorithm [17] to show comparatively better performance over their private implementation of iDistance using 5,000

[2] Publicly available at: http://corpus-texmex.irisa.fr/

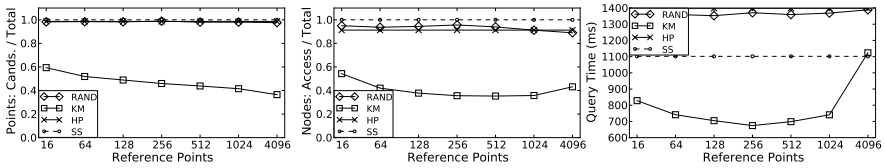

Fig. 13: Results of k-Means and RAND on real data over varying reference points.

reference points. However, without knowledge of algorithmic options, or several baseline experiments to show optimal performance results, we have very little insight into the effectiveness (and reproducibility) of their specific comparison.

In Figure 13, we look at RAND and k-Means over a varying number of reference points, and include HP and SS methods as comparisons. We can see that the only method that performs significantly better than SS is k-Means (KM). Although the number of candidates returned continues to decrease as we add reference points, we can see that after a certain point the overhead costs of additional partitions outweighs the filtering benefits, and the number of nodes accessed begins to increase while query time dramatically rises. We note there exists a clear range of relatively equivalent results from approximately $d/2$ (64) to $4d$ (512) partitions, which might be a combination of many factors including indexing performance and dataset characteristics. This performance plateau also provides an excellent measure for tuning to the proper number of partitions.

We also analyzed the number of partitions that were empty or checked for candidates, and the results of RAND exemplified our concerns over empty partitions and poor reference point placement. Essentially, as the number of random reference points increased, the more empty partitions are created. Worse yet, every non-empty partition is almost always checked due to high overlap and poor placement relative to each other and the data (and query) distributions.

5 Discussion

We see a promising result in Figure 10 at 1000k data points, suggesting that it is still possible to produce better results by moving reference points. This suggests there may exist a more sophisticated solution than the relatively simple methods we presented. We note that because of the default closest distance assignment strategy, when reference points are moved the points assigned to them may change. Thus our efforts to reduce the number of equi-distant points may have been confounded, and if reference points are moved outside the dataspace, their partitions may become empty. Unfortunately, we found no significant difference in results by employing a static partition assignment before and after reference point movement, and therefore did not include the results for discussion. Clearly, more knowledge is required to move reference points in an optimal way that impacts partitioning efficiency. We have begun investigating the idea of *clustering for the sake of indexing*, by learning cluster arrangements explicitly for use as reference points within iDistance [19].

In general, we find a trade-off between dimensionality and dataset size, where more dimensions lead to less precise query regions (classic curse of dimensionality problem), but more points allow smaller regions to fully satisfy a given

query. Space-based methods suffer much worse from dimensionality and are really not ideal for use. We agree with the original authors that $2d$ reference points seems appropriate as a general recommendation. In relatively moderate to small datasets and multi-dimensional spaces, $2d$ is probably overkill but far less burdensome than in exceptionally large datasets and high-dimensional spaces where the cost of additional reference points dramatically increases without providing much benefit. Results strongly support an intelligent data-centric approach to the amount and placement of reference points that results in minimally overlapping and non-empty partitions.

6 Conclusions and Future Work

We presented many complementary results to that of the original iDistance works, and through extensive experiments on various datasets and data characteristics we uncovered many additional findings that were not presented or discussed in prior works. This paper establishes a self-standing baseline for the wide variance in performance of partitioning strategies that opens the door for more directed and concise future works grounded on our findings. These results have also helped to establish an up-to-date benchmark and best practices for using the iDistance algorithm in a fair and efficient manner in application or comparative evaluations.

Many of the results show that traditional iDistance partitions stabalize in performance by accessing entire clusters (within single partitions), despite dataset size and dimensionality. This leads us to explore methods to further segment partitions in future work, so that we can better prune away large sections of dense data clusters. These novel strategies are much like the works of the iMinMax [13] and recently published SIMP [17] algorithms, whereby we can incorporate additional dataspace knowledge at the price of added complexity and performance overhead. Our preliminary work shows potential enhancements to the filtering power of iDistance through novel algorithm extensions that help reduce the negative effects of equi-distant points and partition overlap.

Acknowledgements

This work was supported in part by two NASA Grant Awards: 1) No. NNX09AB03G, and 2) No. NNX11AM13A. A special thanks to all research and manuscript reviewers.

References

1. Aurenhammer, F.: Voronoi diagrams – a survey of a fundamental geometric data structure. ACM Comput. Surv. 23, 345–405 (September 1991)
2. Bayer, R., McCreight, E.M.: Organization and maintenance of large ordered indices. Acta Informatica 1, 173–189 (1972)
3. Beckmann, N., Kriegel, H.P., Schneider, R., Seeger, B.: The R*-tree: an efficient and robust access method for points and rectangles. SIGMOD Rec. 19, 322–331 (May 1990)

4. Bellman, R.: Dynamic Programming. Princeton University Press (1957)
5. Berchtold, S., Bhm, C., Kriegal, H.P.: The pyramid-technique: towards breaking the curse of dimensionality. ACM SIGMOD Record 27(2), pp. 142–153 (1998)
6. Doulkeridis, C., Vlachou, A., Kotidis, Y., Vazirgiannis, M.: Peer-to-peer similarity search in metric spaces. In: VLDB07 (2007)
7. Guttman, A.: R-trees: a dynamic index structure for spatial searching. SIGMOD Rec. 14, 47–57 (June 1984)
8. Ilarri, S., Mena, E., Illarramendi, A.: Location-dependent queries in mobile contexts: Distributed processing using mobile agents. IEEE TMC 5, 1029–1043 (2006)
9. Indyk, P., Motwani, R.: Approximate nearest neighbors: towards removing the curse of dimensionality. In: Proc. of the 30th annual ACM symposium on Theory of computing. pp. 604–613. STOC '98, ACM, New York, NY, USA (1998)
10. 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(2), 364–397 (Jun 2005)
11. Lowe, D.: Object recognition from local scale-invariant features. In: The Proc. of the 7th IEEE Inter. Conf. on Computer Vision. vol. 2, pp. 1150 –1157 vol.2 (1999)
12. MacQueen, J.B.: Some methods for classification and analysis of multivariate observations. In: Cam, L.M.L., Neyman, J. (eds.) Proc. of the fifth Berkeley Symposium on Mathematical Statistics and Probability. vol. 1, pp. 281–297. University of California Press (1967)
13. Ooi, B.C., Tan, K.L., Yu, C., Bressan, S.: Indexing the edges: a simple and yet efficient approach to high-dimensional indexing. In: Proc. of the 19th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of database systems. pp. 166–174. PODS '00, ACM, New York, NY, USA (2000)
14. Qu, L., Chen, Y., Yang, X.: idistance based interactive visual surveillance retrieval algorithm. In: Intelligent Computation Technology and Automation (ICICTA). vol. 1, pp. 71 –75 (oct 2008)
15. Shen, H.T.: Towards effective indexing for very large video sequence database. In: SIGMOD Conference. pp. 730–741 (2005)
16. Shi, Q., Nickerson, B.: Decreasing Radius K-Nearest Neighbor Search Using Mapping-based Indexing Schemes. Tech. rep., University of New Brunswick (2006)
17. Singh, V., Singh, A.K.: Simp: accurate and efficient near neighbor search in high dimensional spaces. In: Proc. of the 15th Inter. Conf. on Extending Database Technology. pp. 492–503. EDBT '12, ACM, New York, NY, USA (2012)
18. Tao, Y., Yi, K., Sheng, C., Kalnis, P.: Quality and efficiency in high dimensional nearest neighbor search. In: Proc. of the 2009 ACM SIGMOD Inter. Conf. on Mgmt. of data. pp. 563–576. SIGMOD '09, ACM, New York, NY, USA (2009)
19. Wylie, T., Schuh, M.A., Sheppard, J., Angryk, R.A.: Cluster analysis for optimal indexing. In: Proc. of the 26th FLAIRS Conf. (2013)
20. Yu, C., Ooi, B.C., Tan, K.L., Jagadish, H.V.: Indexing the Distance: An Efficient Method to KNN Processing. In: VLDB '01: Proc. of the 27th Inter. Conf. on Very Large Data Bases. pp. 421–430. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (2001)
21. 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. pp. 1211–1214. VLDB '06 (2006)
22. Zhang, R., Ooi, B., Tan, K.L.: Making the pyramid technique robust to query types and workloads. In: Proc. 20th Inter. Conf. on Data Eng. pp. 313–324 (2004)
23. Zhang, T., Ramakrishnan, R., Livny, M.: Birch: an efficient data clustering method for very large databases. SIGMOD Rec. 25(2), 103–114 (Jun 1996)