Cluster Analysis for Optimal Indexing

Tim Wylie and **Michael A. Schuh** and **John Sheppard** and **Rafal A. Angryk**
timothy.wylie@cs.montana.edu
Montana State University
Bozeman, MT 59717 USA

Abstract

High-dimensional indexing is an important area of current research, especially for range and kNN queries. This work introduces clustering for the sake of indexing. The goal is to develop new clustering methods designed to optimize the data partitioning for an indexing-specific tree structure instead of finding data distribution-based clusters. We focus on iDistance, a state-of-the-art high-dimensional indexing method, and take a basic approach to solving this new problem. By utilizing spherical clusters in an unsupervised Expectation Maximization algorithm dependent upon local density and cluster overlap, we create a partitioning of the space providing balanced segmentation for a B$^+$-tree. We also look at the novel idea of reclustering for a specific indexing method by taking the output of one clustering method and reclustering it for use in an index. The algorithms are then tested and evaluated based on our error metric and iDistance query performance.

Introduction and Background

Most data mining applications are greatly dependent upon the ability to query and retrieve information from databases quickly and efficiently. To this end, there have been a vast number of indexing schemes proposed for different domains with numerous tunable parameters and their own way of mapping data. Almost no research has focused on finding optimal ways to use some of these indexing methods as well as optimal settings for their parameters based on data dimensionality, the amount of data, etc. For these reasons we seek new methods of optimizing indexing methods based on machine learning techniques.

Efficiently indexing data, especially in higher dimensions, usually involves dimensionality reduction to a well studied data structure such as the B$^+$-tree (Bayer and McCreight 1972). The B$^+$-tree is desirable due to its excellent performance and universal adoption in traditional relational databases. By utilizing the B$^+$-tree for indexing high-dimensional data, these querying methods can be integrated easily into an existing database management system (DBMS). A few of the more successful methods are the Pyramid Technique (Berchtold et al. 1998), iMinMax(θ) (Ooi et al. 2000), and iDistance (Jagadish et al. 2005).

Several efficient high-dimensional indexing systems make use of cluster analysis to partition the data for dimensionality reduction into a B$^+$-tree. For efficiency, the desire is to partition the space or the data in a way that minimizes average query costs (time and data accesses). These methods generally use a well-known algorithm for clustering the data, such as k-means (MacQueen 1967), which is a distribution-based centroid clustering algorithm where spherical clusters are assumed. There are also many variations using constrained or conditional clustering based on k-means (Gordon 1973; Lefkovitch 1980; Wagstaff et al. 2001).

Unlike most clustering techniques, breaking up traditional clusters is often more beneficial for the resulting index due to the lossy dimensionality reduction. Thus, the more uniform the data is, the more evenly distributed the clustering will be, and therefore the tree will be more balanced. The rationale for further splitting the data during indexing to enhance the querying time can be seen in methods like the Extended Pyramid-Technique (Berchtold et al. 1998), the tunable space-partitioning θ line of iMinMax(θ) (Ooi et al. 2000), and the partitioning and reference point placement techniques in iDistance (Jagadish et al. 2005).

For this initial work, we focus on iDistance as the indexing technique to optimize. While iDistance is a state-of-the-art indexing method, little work has been done to try and optimize the method by which the indices are created. The authors of iDistance suggest using k-means with $2d$ partitions (clusters), where d is the number of dimensions, as a general guideline for speed, simplicity, and general effectiveness. While the B$^+$-tree is always balanced, the number of data points within partitions of the tree are usually not.

Cluster analysis is a well-studied field of grouping data by some measure of similarity. However, the presupposition in many algorithms is that there exists a distance measure that implies similarity between instances and the most similar should be grouped. This is often counter-productive to the task of indexing. We desire to keep similar points near each other in the tree, but only if it can be done such that the overall spread of data within the tree is uniform. In a sense, we want to break up standard clustering similarities unless they are beneficial to the data set as a whole, based on our indexing strategy. Given the lossy transformation of data to a single dimension, most nearest neighbor or range queries will generate many false positives because of similar one-

dimensional index values. A better indexing strategy should be able to reduce the total number of candidates in order to reduce the returned false positives and overall page accesses by exploiting the index mapping and decrease the number of candidates with similar index values.

Our approach is a clustering method that reduces overlap and balances the number of points in each B$^+$-tree subtree. This motivation stems from R-trees (Guttman 1984) which were introduced to help create balanced hierarchical trees for indexing more than one dimension. This method of indexing builds the tree while looking at the data. In a similar fashion to our work, it was understood that minimizing overlap should decrease the query time by keeping trees balanced and reducing the number of subtrees to look at for a given query. This is the idea behind R*-trees in optimizing R-trees (Beckmann et al. 1990) when all the data is given initially.

We therefore apply the same idea as a form of unsupervised cluster analysis to build the partitions of iDistance using Expectation Maximization (EM). General EM algorithms (Dempster et al. 1977) and extensions use data distributions to drive the clustering, and thus ignore cluster-balance or spherical overlapping in the space.

We also look at the idea of reclustering by using the converged k-means reference points as the initial seeds to our EM algorithms. Preclustering techniques are common in machine learning, and many have been developed specifically for k-means by giving it seeds that will allow it to converge in a more optimal solution space. Thus, we also want to look at the possibility of adapting existing clustering algorithms to specific indexing problems by reclustering their outputs. This would allow for an existing index to be optimized from a known distribution for better performance in a live system. Clustering for the sake of indexing provides a unique machine learning and data mining problem.

The rest of the paper is structured as follows. Section 2 briefly overviews iDistance, and then Section 3 outlines the problem, the intuition behind our approach, and our error metrics. Section 4 discusses our algorithms, models, parameters, and reclustering. We show the empirical results of clustering and subsequent usage of our methods in Section 5. Finally, Section 6 concludes the paper with our contributions and identifies related future work and possibilities.

iDistance

Most high-dimensional indexing methods use a filter-and-refine strategy. iDistance uses this method by having a lossy transformation to a B$^+$-tree as its index where a point is assigned to a partition. When processing a query, it uses the mapping function to find the areas of the tree (partitions) to search sequentially, and these results are the candidate set. Then the algorithm refines the set by removing points that are not actually within the query sphere in the original high-dimensional space.

iDistance assumes a spherical partitioning of the data-space. At the center of each sphere, or partition, is the reference point. The indexing method tracks these reference points and the radius of each sphere. Points can be assigned to partitions in a number of ways. The most common method is point assignment to the nearest reference point, which creates a Voronoi tessellation of the space.

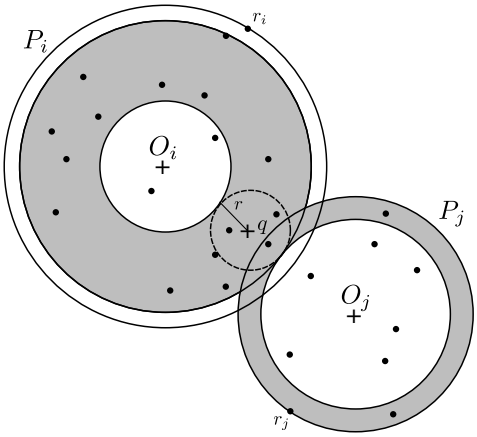


Figure 1: A basic example of iDistance indexing partitions with a sample query q of radius r. Partitions P_i, P_j with reference points and radii O_i,O_j and r_i,r_j, respectively.

The B$^+$-tree mapping is straightforward. The index value for point p in partition P_i with reference point O_i is $y_p = i*c+dist(O_i, p)$ where c is a constant for spacing partitions in the tree to ensure that an index value for any point p in partition P_i will not conflict with values in another partition P_j s.t. $i \neq j$.

Figure 1 shows a two-dimensional example of two partitions, a query sphere, and the shaded areas that must be searched due to the lossy mapping. We see the query sphere is inside P_i and intersecting P_j. Due to the lossy transformation, the overlap of P_j requires that we search through every point in partition P_j that is the same distance from the reference point O_j. Thus, the more the partition spheres overlap, the more false positives the algorithm must look through that are outside the query sphere. See (Jagadish et al. 2005) for a thorough explanation of iDistance.

Index Clustering

High-dimensional data is commonly indexed in a lossy one-dimensional transformation and stored in a B$^+$-tree for efficiency and quick retrieval. These indexing methods do this by dividing up the high-dimensional space using some form of space-based or data-based partitioning. However, due to the unavoidable lossy transformation, often this results in the partitions varying greatly in the amount of data in each subtree, and many points being mapped to equivalent values that results in more candidates. Unbalanced trees result in unpredictable performance when querying the data, and overlap often leads to more false positive candidates. Given we have all of our data before indexing, we need an efficient and fast method of partitioning the data into clusters to optimize the tree structure for query processing and reduce the number of points mapped to the same index values.

Error Metrics

Given N is the number of data points and k is the number of desired clusters. A spherical cluster P_i, where $1 \leq i \leq k$, has a centroid O_i, a radius r_i, and a population p_i, which is

simply the number of elements assigned to that cluster. We look at two different error metrics.

Cluster Overlap We want to minimize the amount of overlap between spheres. Rather than use volume, we can calculate the amount they overlap along a vector as shown in Equation 1. Further, we set the function to zero if they do not overlap in order to avoid negative values.

$$\mathcal{O}_{i,j} = \begin{cases} (r_i + r_j) - dist(O_i, O_j), & \text{if } P_i, P_j \text{ overlap} \\ 0, & \text{otherwise} \end{cases} \quad (1)$$

Equation 2 calculates the average percentage of overlap for all clusters. Given k clusters there are $\binom{k}{2}$ unique pairwise combinations which simplifies to $C = k(k-1)$ in our equation, and we multiply by two in order to compare overlap based on the diameter and not the radius. Unfortunately, when averaging for all possible combinations, E_O results in values that are often too small to be useful. Therefore, we calculate only the average overlap of the diameter of spheres that are overlapping. Let $V_o = \{\overrightarrow{O_i O_j} | \mathcal{O}_{i,j} > 0\}$ which are the vectors between all centroids of overlapping spheres, and then $C = |V_o|$.

$$E_{\mathcal{O}} = \frac{1}{2C} \sum_{i=1}^{k} \sum_{\substack{j=1 \\ j \neq i}}^{k} \frac{\mathcal{O}_{i,j}}{r_i} \quad (2)$$

Using all combinations has one drawback: the error could increase after an update separates two spheres that were previously overlapping. If the overlap was small such that dividing by $|V_o^t|$, at time t, with the extra term was causing the error to be small, and then after the update $|V_o^t| = |V_o^{t-1}| - 1$, and the error could actually go up. This is likely to be rare since the overlap would be extremely small, but it should be further investigated in the future.

Partition Balance (Population) The second measure we look at is the population difference. We want this to be zero between all clusters, but this may be impossible in some instances without severe overlap as a trade-off. Equation 3 shows the normalized population difference, which is the percentage of population imbalance to the desired amount in the cluster: N/k should be the approximate number of points per cluster for an even distribution. Thus, we need to minimize the average relative population error per cluster, shown in Equation 4.

$$P_{i,j} = \frac{|p_i - p_j|}{N/k} \quad (3) \qquad E_{\mathcal{P}} = \frac{1}{k} \sum_{i=1}^{k} \frac{|p_i - N/k|}{N/k} \quad (4)$$

We view these errors as part of the same clustering system and combine them as a vector in two dimensions. The total error vector is $\overrightarrow{E} = E_{\mathcal{O}} + E_{\mathcal{P}}$, and thus the value we minimize for our system is the magnitude of the combined error vector. This means the error is between 0 and $\sqrt{2} \approx 1.4$.

$$Error = ||\overrightarrow{E}|| = \sqrt{E_{\mathcal{O}}^2 + E_{\mathcal{P}}^2} \quad (5)$$

EM Algorithm

An Expectation Maximization (EM) algorithm usually has an assignment and an update step. The assignment phase assigns each point to a cluster based on a defined measure or method. The update step then uses the associations of points and clusters to update the cluster positions. The two steps iterate until some stopping condition has been met. Based on the error metrics defined, we create three variations of the EM algorithm designed to minimize the overlap of the cluster spheres and simultaneously minimize the population imbalance. All three algorithms have the same update function, but the assignments are different.

Assignment

One way to ensure cluster balance is to enforce it during the assignment phase of the EM algorithm. The first two methods attempt this and the third assigns points greedily by cluster coverage.

Algorithm 1 (A1) Look at each point and assign it to the nearest cluster centroid as long as the cluster has less than N/k points in it. If it is full, attempt to assign to the next closest, and so on, until it is assigned. If all clusters are full, ignore the population constraint and assign it to the closest. This algorithm is dependent on the ordering of the points used for assignment, but also makes the reference points move more to decrease this dependency, i.e. the arbitrary point arrangement is used to help partition the space.

Algorithm 2 (A2) For each cluster centroid, rank the N/k nearest neighbors. Greedily assign each point to the cluster where it has the highest rank. Points that are not in any of the rankings are assigned to the nearest cluster by default. This method of assignment is designed to give each reference point the optimal nearest neighbors, but gives a bias to the ordering of reference points.

Algorithm 3 (A3) First, assign all points that are only in one cluster sphere. Next, assign points that are no longer covered by a sphere after the update; assign point p to the nearest partition P_i and set $r_i = dist(O_i, p)$. Finally, assign all points that are in the overlap of multiple spheres. Greedily assign each point to the encompassing sphere with the smallest population. Here, we have an algorithm that must adjust the size of the cluster spheres as well as move them to fix the population imbalance.

Update

The update step is the same for all three algorithms, and is shown in Equation 6. The basic premise of the first term is to calculate the overlap of the two spheres and move them apart to eliminate the overlap. However, since we also want equal populations, the second term moves the spheres closer if the population imbalance is too great.

$$\overrightarrow{O_i} = \overrightarrow{O_i} - \sum_{\substack{j=1 \\ j \neq i}}^{k} \overrightarrow{O_i O_j} (\omega \mathcal{O}_{i,j} - \lambda \mathcal{P}_{i,j}) \quad (6)$$

Equation 6 updates cluster centroid O_i based on another cluster (partition) P_j. There are two hyperparameters, ω and

λ, which weight the amount of influence the overlap and population components have on the update. Extensive testing has shown that, for our error metric and application, setting each hyperparameter equal to one is optimal, but other applications may yield better results with different weights.

Model Parameters

There are many hyperparameters in the model. These additional hyperparameters were set based on empirical performance and ensuring appropriate convergence.

k-clusters The number of clusters is set to d (the number of dimensions) based on balancing the number of clusters in a given dimension for indexing. There are justifications for basing the number of partitions on the dimensionality in iDistance (Jagadish et al. 2005), but there are also good arguments to base k on the number of data points. Looking at the balance between dimensions, the number of points, and the number of clusters is left for future work.

Max radius A maximum radius restriction of 0.5 was imposed during the update step to keep clusters from covering the entire data space. It may be more appropriate for the restriction to be based on the dimension because in higher dimensions it creates smaller clusters and could impose an unintended limit on the flexibility of the algorithm.

Convergence window The model converges based on when the error stops decreasing on average in a sliding "window", where a set number of iterations of the model are saved, and the best iteration within the window is returned. The size of the sliding window was set to five iterations.

Max distance Occasionally during the update step, a centroid can be moved too far outside the data space. Thus, a maximum distance of $2\sqrt{d}$ from the center of the data space is imposed, which must be less than the spacing constant c in iDistance.

Radius normalization Along with the max distance, orphaned centroids pushed outside the data space might not be assigned any points. Thus, to encourage clusters with fewer points to get more, and to discourage too many points, the radius is normalized based on population, $r_i = r_i * (N/k)/(p_i + 1)$. One is added to the population to ensure it is defined, and the maximum radius restriction is also enforced if this value is too large.

Reclustering

Along with the standard iDistance k-means-based partitions and our three algorithms, we also look at using the output of k-means as the seed for A1, A2, and A3, which are listed as KMA1, KMA2, and KMA3 in the results. This demonstrates how k-means could work with our methods for creating balanced space partitioning. Thus, in this context our algorithms can be thought of as a post-processing procedure for k-means that try to optimize it for iDistance.We are reclustering by taking existing clusters and tuning them for better performance within a specific application.

Experimental Results

Setup

For the evaluation of our algorithms we take a slightly different approach to the general clustering literature since our goals are different. We look at the clustering results based on our error metrics of what should make balanced, well-performing B^+-trees, and then we load the clustered results into iDistance, run extensive kNN queries, and report the total candidates and nodes accessed on average. The tree statistics are similar for B^+-trees generated from k-means or our method due to them having the same number of points, however, the subtrees for partitions are vastly different. iDistance generally uses k-means based partitions, thus all results for the standard iDistance algorithm are denoted as 'KM' in the results.

Data Sets

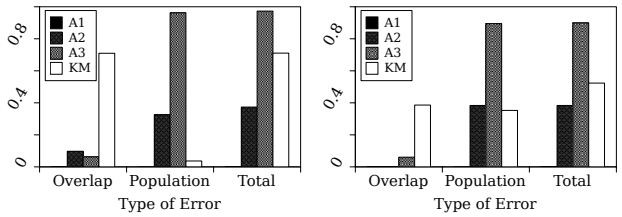
Given the purpose and application of this investigation, the data sets used are all synthetic in order to explore specific aspects of the algorithms as they compare to k-means and each other. We use both clustered and uniform data sets. The clustered sets each consist of 12 Gaussian clusters. The first has tight clusters with a standard deviation (std dev) of 0.1 and the second has looser clusters of 0.2 std dev within the unit space. Each data set contains 10,000 points, and these were created in 4, 8, 16, and 32 dimensions. However, results are only shown in 16 and 32 as the other results are less interesting. In lower dimensions we do much better, but since iDistance was designed for higher dimensions, we focus on the highest ones we evaluate.

Model Error Results

We first look at how the algorithms performed on average using our error metric. We look at an average of 10 runs for all algorithms and in all dimensions for both uniform and clustered data. In Figure 2 we show the overlap, population, and total error for both uniform and clustered (0.2 std dev) data for the $16d$ runs – the other dimensions show similar results. As expected, A1 and A2 have nearly zero error for population and also little overlap. Algorithm A1 has zero error in most of the models because the points can be equally distributed in such a way that the clusters do not overlap, so A1 is not visible in the figures.

k-means is known to prefer similar sized clusters in many cases, and this is also reflected in Figure 2(a). In uniform data, k-means has almost no population error and only overlap error, but in highly clustered data, k-means no longer maintains this balance. As expected, the first two algorithms based on reducing overlap and population imbalance have lower total error than k-means.

Another note of interest is the time and number of iterations of each algorithm. A1, A2, and A3 usually had less than 10-20 iterations. k-means, however, had greater than 100 iterations, but the error (based on our metric) did not change significantly after the 4th or 5th iteration. This may indicate that k-means may be just as effective for indexing within iDistance if stopped early based on our metric rather than the traditional stopping criteria.



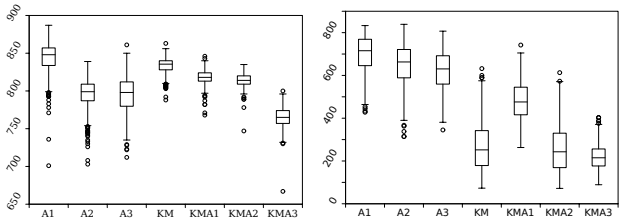
(a) Uniform (b) Clustered

Figure 2: Model error for uniform and clustered data in 16d.

iDistance Results

Using the runs from the error results, the best model for each algorithm was chosen based on our error metric. The clustered results were then used as partitions for iDistance with the corresponding data set for the following experiments.

Nodes Accessed iDistance is optimized for nearest-neighbor searches, so we base our results on kNN queries. For each method, a kNN query for 10 neighbors (k=10) was performed on 500 random points in the space. Figure 3(a) shows these statistics for nodes accessed for the algorithms in uniform data. We can see that k-means has a higher average number of nodes, and that algorithms A2 and A3 outperform it on average. Reclustering k-means using our algorithms also improves the average performance. However, despite A1 having the lowest error, it has the worst performance. This demonstrates our error metric alone is not indicative of iDistance performance.

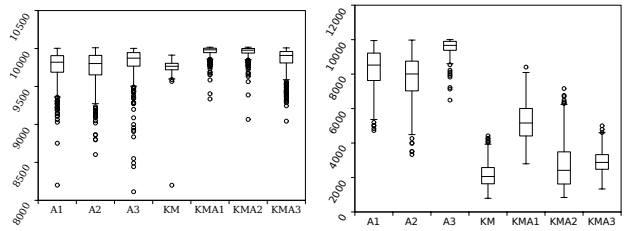


(a) 16d Uniform Nodes (b) 16d Clustered Nodes

Figure 3: Average nodes per query over 500 10NN queries in 16 dimensions for (a) uniform and (b) clustered data.

Figure 3(b) shows the statistics for clustered data, and this is where k-means has an advantage since the data distributions can be exploited. Here, only the reclustered versions are comparable. KMA3 has drastically reduced the spread of k-means and outperforms all the other algorithms.

Candidate Sets We now examine the candidates accessed. For each query only the 10 nearest neighbors are desired, meaning any others are false positives. The goal is to reduce the number of false positives by improving the clustering layout in the data space. Figure 4 shows these results for uniform and clustered data in 16d. Here, we see results similar to the accessed nodes. With uniform data, our algorithms have a wider spread, but a similar average to k-means. With clustered data, k-means outperforms all methods, and only those using the k-means centroids as seeds are comparable.



(a) 16d Uniform Candidates (b) 16d Clustered Candidates

Figure 4: Average candidates per query over 500 10NN queries in 16 dimensions for (a) uniform and (b) clustered data.

Figure 5 also shows some favorable results in 32d on clustered data. The reclustered algorithms actually outperform k-means with the exception of KMA3, yet KMA3S does well. KMA3S is a form of Expectation Conditional Maximization (ECM) (Meng and Rubin 1993). During the update phase, the centroids are fixed and each centroid is updated independently based on the fixed value of the others. This was implemented and tested with all of the algorithms, but there were no significant differences in most of the experimental results and thus were omitted from the other graphs.

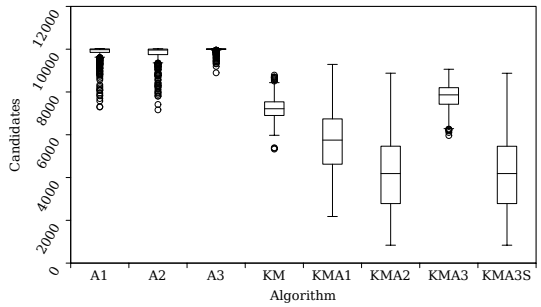


Figure 5: Clustered average candidates per query over 500 10NN queries in 32 dimensions.

kNN Size Most of the clustered results show that generally k-means performs well, but this is partly dependent on kNN size, how tight the clusters are, and the dimension of the space. Results vary greatly when these are modified.

In 16 dimensions with clustered data, we can see in Figure 6 how the algorithms tend toward selecting all the candidates as the size of k grows. What is also interesting about this graph is that KMA3 outperforms k-means, and even when selecting 10% of the data points, it is still only looking at about 75% of all the candidates. This provides some validity to the idea of reclustering existing clusters. A similar result was also witnessed in 8 and 32 dimensions. This also demonstrates that given the right seed, our algorithms can compete in clustered data with k-means.

In general, as the number of nearest neighbors increases, our lossy transform will require more false candidates to be examined. This query selectivity problem makes large kNN queries behave similar to uniform data. Given our algorithms perform better in uniform data, the larger kNN queries are beneficial to compare with.

An advantage k-means has in tightly clustered data, is that the distribution is already minimal in overlap, but the clus-

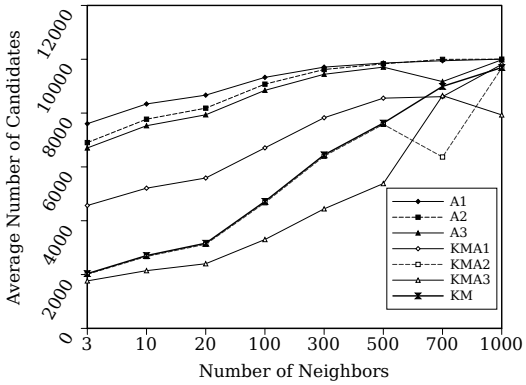


Figure 6: The affect on the average number of candidates seen when varying the size of the kNN search in clustered 16 dimensional data.

ters can still have a larger radius, which our models limit. Our algorithms, while more balanced in the tree structure, suffer in the clustered data because with an even population amongst the partitions, we will always have approximately the same average number of candidates to look through. k-means, however, will have fewer false candidates mapped to the same area of the tree given the clusters are unbalanced. The hope was that our methods would use points from multiple clusters, but it is more likely that our spheres settled in the densest parts of clusters whenever possible in order to evenly distribute the points.

Conclusion

In this paper we have introduced the idea of clustering for the sake of indexing, and then presented a novel approach of applying an unsupervised EM algorithm to cluster and recluster data for iDistance. This focuses on the use of cluster analysis to generate index trees for a specific indexing method, as opposed to traditional, purely similarity-based, clustering. We have shown that our basic algorithms designed for iDistance perform well with uniform data. Further, we have shown that using our algorithms to recluster k-means can yield equal or better performance benefits in certain situations. Thus, existing clusters can be adapted (reclustered) quickly for better use in specific indexing techniques. Our methods alone provided no significant improvements for iDistance over k-means in clustered data, and in most cases performed worse. As indicated by the reclustered versions though, this is largely due to random seeding, and a preclustering algorithm would greatly improve this.

Our error metric was not as indicative of a correlation to the query candidates and nodes as we would have hoped. More work needs to be done to understand the balance of overlap, population, and the specific iDistance index mapping function. Our results are encouraging, and demonstrate the need for index-specific clustering techniques, as well as some positive results and open questions in that direction. Below we list several of the unanswered questions raised by our study for future work on this topic.

(1) Would other clustering algorithms, used as seeds, yield better results for our purposes? We tested against k-means due to standard practices, but other methods may be better.

(2) For our algorithms, we need to test changing the radius for clusters in higher dimensions. We know that some of k-means's effectiveness was due to not having this restriction.

(3) Other hyperparameters, such as the number of clusters for a given dimension, also need more thorough experimentation. How will an adaptive number of clusters based on dimension and dataset size change the results, and what is the best way to determine this trade-off?

(4) Will weighting the hyperparameters unevenly allow for better performance in clustered data? In general, can our error metric be adapted to more accurately reflect the iDistance mapping function?

(5) Can better clustering methods be designed for iDistance and consistently improve average retrieval?

(6) Can we create similar error metrics and clustering algorithms for indexing methods other than iDistance?

References

Bayer, R., and McCreight, E. 1972. Organization and maintenance of large ordered indices. *Acta Inform.* 1:173–189.

Beckmann, N.; Kriegel, H.-P.; Schneider, R.; and Seeger, B. 1990. The R*-tree: an efficient and robust access method for points and rectangles. *SIGMOD Rec.* 19(2):322–331.

Berchtold, S.; Bhm, C.; and Kriegal, H.-P. 1998. The pyramid-technique: towards breaking the curse of dimensionality. *ACM SIGMOD Rec.* 27(2):142–153.

Dempster, A. P.; Laird, N. M.; and Rubin, D. B. 1977. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society, Series B* 39(1):1–38.

Gordon, A. D. 1973. Classification in the presence of constraints. *Biometrics* 29(4):pp. 821–827.

Guttman, A. 1984. R-trees: a dynamic index structure for spatial searching. *SIGMOD Rec.* 14:47–57.

Jagadish, H. V.; Ooi, B. C.; Tan, K.-L.; Yu, C.; and Zhang, R. 2005. iDistance: An adaptive B+-tree based indexing method for nearest neighbor search. *ACM Trans. Database Syst.* 30:364–397.

Lefkovitch, L. P. 1980. Conditional clustering. *Biometrics* 36(1):pp. 43–58.

MacQueen, J. B. 1967. Some methods for classification and analysis of multivariate observations. In Cam, L. M. L., and Neyman, J., eds., *Proc. of the 5th Berkeley Sym. on Math. Stat. and Prob.*, volume 1, 281–297. UC Press.

Meng, X.-L., and Rubin, D. B. 1993. Maximum likelihood estimation via the ECM algorithm: A general framework. *Biometrika* 80(2):267–278.

Ooi, B. C.; Tan, K.-L.; Yu, C.; and Bressan, S. 2000. Indexing the edges: a simple and yet efficient approach to high-dimensional indexing. In *Proc. of the 19th ACM SIGMOD-SIGACT-SIGART Sym. on Prin. of Database Syst.*, PODS '00, 166–174. New York, NY, USA: ACM.

Wagstaff, K.; Cardie, C.; Rogers, S.; and Schroedl, S. 2001. Constrained k-means clustering with background knowledge. In *ICML*, 577–584. Morgan Kaufmann.