Evaluation of the iMinMax Algorithm for Indexing and Retrieval with High Dimensionality

Hilarion Lynn,[1] Cole Schock,[2] and Liessman Sturlaugson[3]
Department of Computer Science, Montana State University
Bozeman, Montana 59717-3880
[1]hilarionl@gmail.com
[2]dcolesch@gmail.com
[3]liessman.sturlaugson@cs.montana.edu

***Abstract*—We have implemented and evaluated the iMinMax algorithm with point, range, and k-nearest neighbor queries on uniform, clustered, and real-life datasets of varying sizes. We have benchmarked the algorithm against sequential scan algorithms for each of the three query types. Our experiments show that iMinMax offers improvements over sequential scan, especially as the data size and number of dimensions increase. Lastly, we provide an extension for iMinMax, showing how to use the median of the data to approximate an optimal θ, the parameter that accounts for skewed data distributions that would otherwise degrade iMinMax performance.**

I. Introduction

Many efficient indexing structures exist for storing and querying low-dimensional data. The B^+-tree offers low cost insert, delete and search operations for single dimensional data. The R tree extends the concepts of the B^+-tree to 2 or more dimensions by inserting minimum bounding rectangles into the keys of the tree. But viewing the data space over a set of features usually causes the dimensionality of the dataset to substantially increase.

Unfortunately, the performance of all traditional indexing techniques deteriorates when employed to handle highly dimensional data. On data above 8 dimensions, most of these techniques perform worse than a sequential scan of the data. This performance degradation has come to be called the "curse of dimensionality" [3]. Improved techniques for indexing highly dimensional data are necessary.

A recent trend in addressing the problem of highly dimensional data is to employ an algorithm to map the multi-dimensional values of a high dimensional record to a single dimensional index [9]. After the data is collapsed to a single dimensional index, it is possible to re-use existing algorithms and data structures that are optimized for handling single dimensional data, such as the B^+-tree. Examples of this strategy are the Pyramid-Technique [2], the iMinMax algorithm [6], and the iDistance algorithm [10]. We have implemented and benchmarked one of these strategies, the iMinMax algorithm, and demonstrated that it is resistant to the "curse of dimensionality."

II. Related Work

Mapping to a single dimension from any higher dimensions will always result in a lossy transformation. Therefore, all of these techniques must employ a filter and refine strategy. To be useful, the transformation should allow much of the data to be ignored for a given query. At the same time, the transformation must ensure that the filter step misses no true positives, so that, after the refine step removes the false positives, the result is *exactly* the points that match the query.

One approach for mapping to one dimension, the Pyramid-Technique [2], partitions a data space of d dimensions into $2d$ hyper-pyramids, with the top of each pyramid meeting at the center of the data space. The index of each point has an integer part and a decimal part. The integer part refers to the pyramid in which the point can found, and the decimal part refers to the "height" of the point within that pyramid. For range queries, the algorithm calculates all the pyramids that the range intersects and searches the appropriate interval along the height within the pyramid. Although two points may be relatively far apart in the data space, because each point is indexed by a single real value, any number of points can potentially be mapped to the same index. This is where the Pyramid-Technique must use a filter and refine strategy, generating a candidate set that guarantees the inclusion of all true positives and then using the full feature vector of each point to remove the false positives.

Following a similar strategy, the iMinMax algorithm, first presented in [6] and evaluated here, maps each point to the "closest edge" instead of explicitly partitioning the data space into pyramids. By mapping to axes instead of pyramids, they reduce the number of partitions from $2d$ to d. The simple mapping function was also intended to avoid more costly pyramid intersection calculations. The iMinMax algorithm has also been combined with the VA-file for an approximate indexing algorithm [8] by first applying the VA-file approach for dimensionality reduction.

While designed with point and range queries in mind, the original implementations of the Pyramid-Technique and the iMinMax algorithm did not explicitly address k-nearest neighbor (KNN) queries. On the other hand, the original iDistance algorithm [10], later published with additional experiments [5], was specifically optimized for KNN queries. In this technique, a set of "reference points" are first placed throughout the data space. The algorithm then indexes each point based on its distance to the nearest reference point.

$$q_j = \begin{cases} [j + \max_{i=1}^{d} x_{il}, j + x_{jh}] & \text{if } \min_{i=1}^{d} x_{il} + \theta \geq 1 - \max_{i=1}^{d} x_{il} \\ [j + x_{jl}, j + \min_{i=1}^{d} x_{ih}] & \text{if } \min_{i=1}^{d} x_{ih} + \theta < 1 - \max_{i=1}^{d} x_{ih} \\ [j + x_{jl}, j + x_{jh}] & \text{otherwise} \end{cases} \quad (2)$$

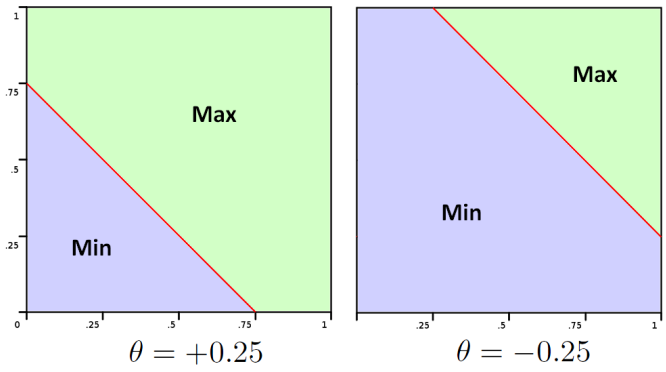

Fig. 1. Effect of changing θ to ± 0.25

III. The iMinMax Algorithm

We now describe the iMinMax algorithm in more detail, along with its extension to KNN developed in [7].

We assume the data is normalized such that each data point x resides in a unit d-dimensional space. A data point x is denoted as $x = (x_1, x_2, \ldots, x_d)$ where $x_i \in [0, 1)\ \forall i$. Let $x_{max} = \max_{i=1}^{d} x_i$ and $x_{min} = \min_{i=1}^{d} x_i$. Each point is mapped to a single dimensional index value $f(x)$ as follows:

$$f(x) = \begin{cases} d_{min} + x_{min}, & \text{if } x_{min} + \theta < 1 - x_{max} \\ d_{max} + x_{max}, & \text{otherwise} \end{cases} \quad (1)$$

The parameter θ can be tuned to account for skewed data distributions in which much of the data would otherwise be mapped to the same edge, resulting in a less efficient search through the B$^+$-tree. In the simple case when $\theta = 0$, each point is mapped to the axis of the closest edge and is appropriate for uniformly distributed data. When $\theta > 0$, the mapping function is biased toward the axis of the maximum value, while $\theta < 0$ biases it toward the axis of the minimum value. Fig. 1 shows how changes to θ of ± 0.25 influence the data to map toward the Min or Max edge.

Range queries are first transformed into d one-dimensional subqueries. The range query interval for the jth dimension, denoted q_j, is calculated by Equation 2. The variables x_{il} and x_{ih} represent the low and high bound, respectively, for the range interval in the ith dimension. In the original iMinMax paper [6], they prove that the union of the results from the d subqueries is guaranteed to return the set of all points found within the range, while no smaller interval on the subqueries can guarantee this. Moreover, they prove that at most d subqueries must be performed. In fact, they prove that a subquery $q_i = [l_i, h_i]$ need not be evaluated if one of the following holds:

$$(i)\ \min_{j=1}^{d} x_{jl} + \theta \geq 1 - \max_{j=1}^{d} x_{jl} \text{ and } h_i < \max_{j=1}^{d} x_{jl}$$

$$(ii)\ \min_{j=1}^{d} x_{jh} + \theta < 1 - \max_{j=1}^{d} x_{jh} \text{ and } l_i > \min_{j=1}^{d} x_{jh}$$

This occurs when the all the answers for a given query are along either the Max edge (i) or the Min edge (ii). Thus if either of these conditions hold, the answer set for q_i is guaranteed to be empty and thus can be ignored.

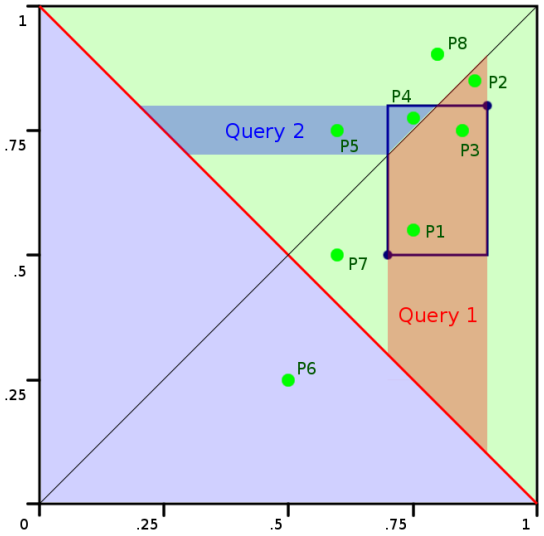

Fig. 2. Example iMinMax range query in 2 dimensions with $\theta = 0$

Fig. 2 shows the two subqueries generated by an example range query in 2 dimensions when $\theta = 0$. Query 1 returns $\{P1, P2, P3\}$ during the filter step and then refines by removing $P2$. Likewise, Query 2 returns $\{P4, P5\}$ and then refines by removing $P5$.

The original iMinMax paper did not address KNN queries. A naïve approach that reuses the range query would be to just initialize a small range query around the query point and slowly increase the range window until at least k points are found and then sort them by distance. However, choices of the initial range size and the increment for resizing the range could be inefficient depending on the data distribution and where the query point falls within that distribution.

Shi and Nickerson [7] proposed a KNN extension for iMinMax that is intended to improve upon the naïve approach by using a decreasing radius search strategy. The index of the query point q is first calculated using Equation 1. The leaf containing the closest value to the index is then found. The leaves to the left and right are then searched as necessary to

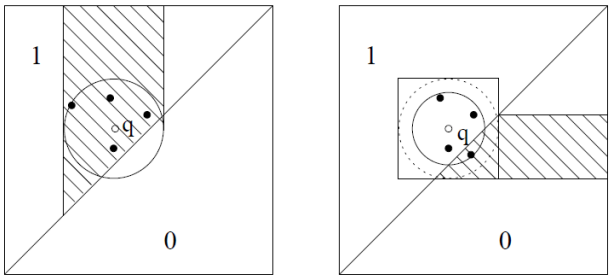

Fig. 3. Example from [7] of decreasing radius with $k = 4$ and $\theta = -1$

find an initial candidate set of k neighbors. They note that, for iMinMax, if a point v is mapped to the same axis as the query point, the difference between their iMinMax index values can be no greater than the Euclidean distance between the points. After finding a set of k candidates, the candidate point furthest from the query point defines a range query. A subquery defined by that range is then performed on a remaining dimension. If a closer point is found, it replaces the current furthest point, and the range is reduced to the new furthest point. This process continues until all subqueries checking each dimension have been performed. The benefit of this algorithm is that the range never increases after the initial k candidates are found and that each subquery has the potential of reducing the query range even further. Fig. 3 shows an example of the algorithm decreasing the radius. After the initial 4 neighbors are found in dimension 1, the algorithm searches the range in dimension 0. Upon finding a closer neighbor, the range shrinks to the distance of the new furthest point.

IV. Datasets

As [4] points outs, the efficiency of an index-based querying algorithm depends on the number of dimensions, the number of points, and on the data distribution. The iMinMax algorithm was tested on 27 variations of three underlying datasets. Each dataset had three versions corresponding to increasing dimensionality: 16, 64, and 256 dimensions. Each dataset also had three versions corresponding to increasing number of data points. Lastly, each dataset came from one of three underlying distributions: uniform, clustered, and real-life datasets. Each dataset maintained four decimal place precision in each dimension for each point.

A. Uniform

The three sizes from the uniform dataset were 1K, 10K, and 100K. For each dataset size and for each dimensionality (16, 64, and 256), the points were placed uniformly randomly inside a unit hypercube.

B. Clustered

The three sizes for the clustered datasets were also 1K, 10K, and 100K. Each clustered dataset used 10 clusters. For each dataset size and for each dimensionality (16, 64, and 256), the locations of the 10 clusters were randomized.

C. Real-Life

The real-life dataset came from NASA's TRACE mission images. The three sizes for the clustered datasets were 160, 1600, and 16,000. For each dataset size and for each dimensionality (16, 64, and 256), the points were chosen randomly from the real-life dataset.

V. Experiments

The iMinMax(θ) algorithms for point, range, and KNN queries were implemented in C++. The keys of the B^+-tree were the real-valued indices returned by iMinMax, while the values associated with the keys were indices of the actual points in an array stored in memory. The source code and wiki for this project can be found at http://code.google.com/p/iminmax. More up-to-date but perhaps less stable source code can be found on the development server repository at http://edu.tyob.info/viewgit/.

The algorithms were benchmarked using a machine running Debian 6.0.3 with a 1.6 GHz Intel Atom Dual Core N330 processor and 2 GB DDR2 RAM.

VI. Results

This section summarizes the results of the experiments, including the overhead of tree and index construction. The performance of iMinMax is compared against sequential scan for point, range, and KNN queries. Typically, each type of query for each dataset had two queries, and so the performance was averaged over these two queries. Although the performance measures in the code includes counting the number of nodes accessed as well as the average search time, the shapes of the curves were found similar enough that only the average query execution time is reported here as representative of the algorithms' performance.

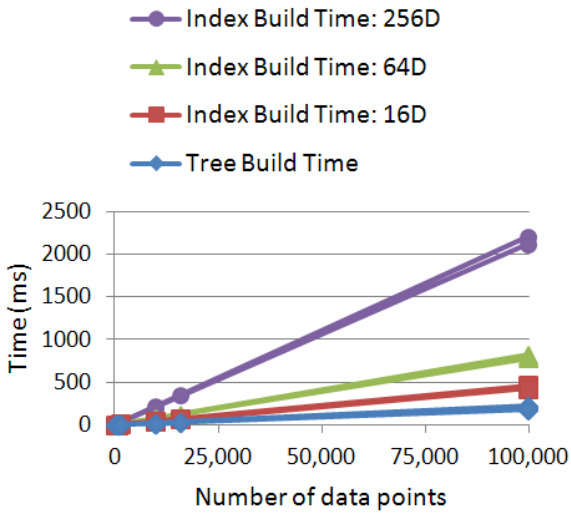

Fig. 4. iMinMax index and tree construction times

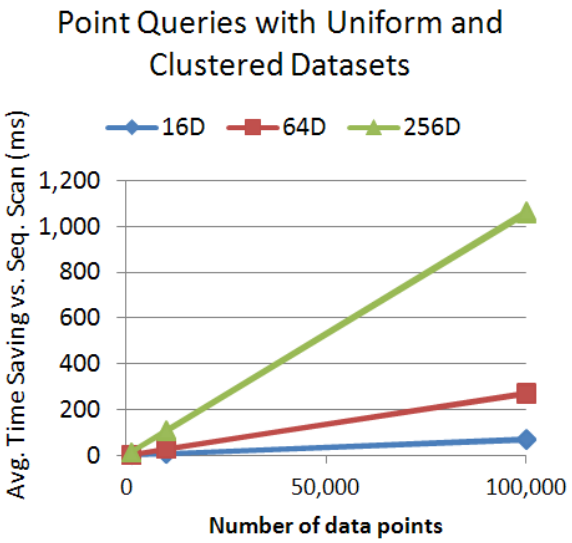

Fig. 5. Average time savings of iMinMax over sequential scan for point queries with uniform and clustered datasets

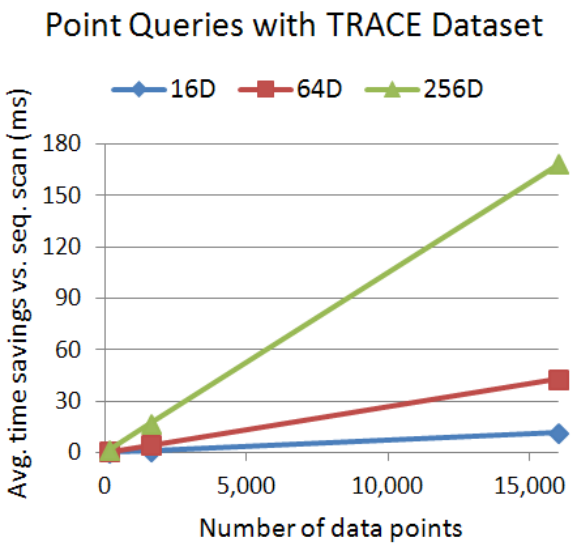

Fig. 6. Average time savings of iMinMax over sequential scan for point queries with TRACE dataset

A. Index and Tree Construction

Fig. 4 shows how the size of the data and its dimensionality affected the index and B^+-tree construction times for all 27 datasets. The index build times include calculating each point's iMinMax index value, as well as the insertion of the key/value pair into the B^+-tree. Tree build time only includes the tree insertion times, not the iMinMax index creation times.

The B^+-trees needed from 70 to 76 nodes (a tree height of 3 levels) to store the indices to 1K data points, from 720 to 738 nodes (4 levels) to store 10K, and from 7248 to 7662 (5 levels) to store 100K. The average fill (number of non-empty slots in a node) for each of the trees was $70 \pm 6\%$.

B. Point Query

The performance of iMinMax and sequential scan was averaged over the two point queries available for each dataset. Fig. 5 shows the relative performance of the iMinMax point query versus the sequential scan with the uniform and clustered datasets, and Fig. 6 shows the relative performance of the iMinMax point query versus the sequential scan with the TRACE dataset.

C. Range Query

For the range query, both "narrow" and "wide" ranges were tested with two queries each. The performance difference between the "wide" and "narrow" ranges was negligible, and so these were averaged together for each dataset. Fig. 7 shows the relative performance of the iMinMax range query versus the sequential scan with the uniform and clustered datasets, and Fig. 8 shows the relative performance of the iMinMax range query versus the sequential scan with the TRACE dataset.

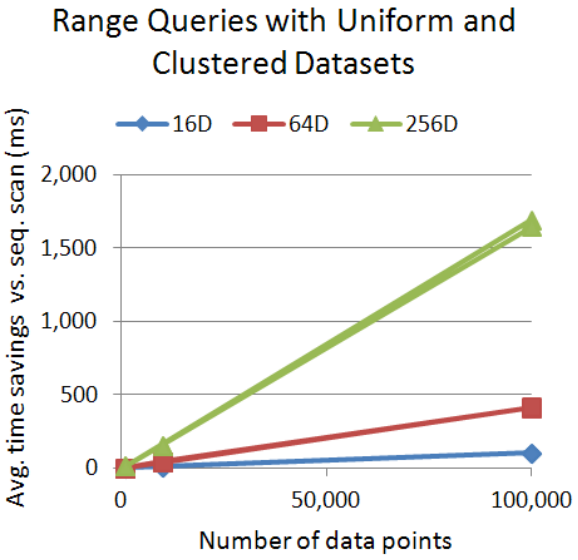

Fig. 7. Average time savings of iMinMax over sequential scan for range queries with uniform and clustered datasets

D. K-Nearest Neighbors Query

The performance of iMinMax and sequential scan was averaged over the two KNN queries available for each dataset with $k = 3$ for all queries. Fig. 9 shows the relative performance of the KNN algorithm versus the sequential scan with the uniform and clustered datasets, and Fig. 10 shows the relative performance of the KNN algorithm versus the sequential scan with the TRACE dataset.

VII. Discussion

The index and tree build times of Fig. 4 show that the additional data structures and preprocessing required by the iMinMax algorithm are not prohibitively expensive and scale

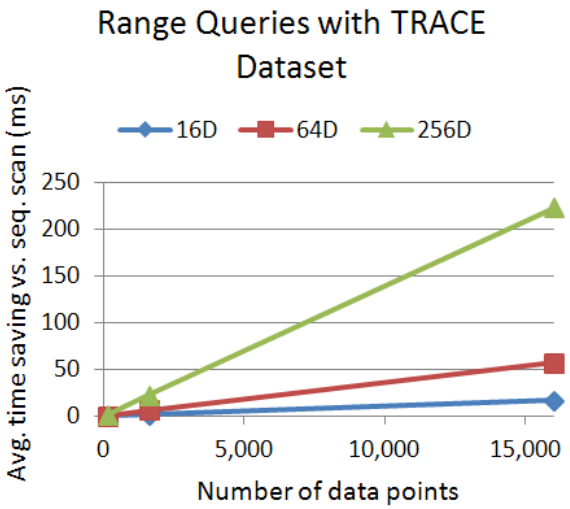

Fig. 8. Average time savings of iMinMax over sequential scan for range queries with TRACE dataset

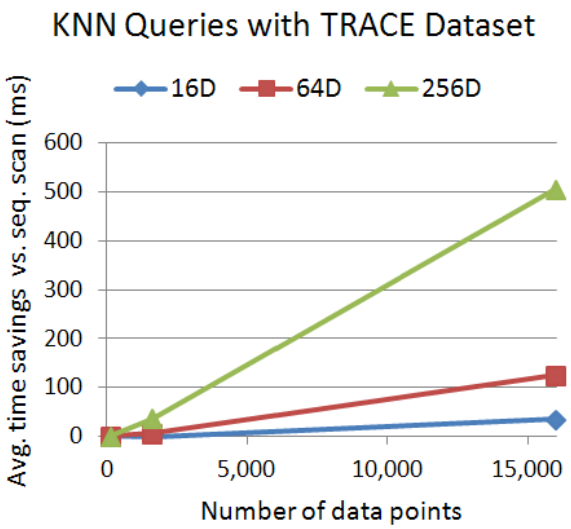

Fig. 10. Average time savings of iMinMax over sequential scan for KNN query ($k = 3$) with TRACE dataset

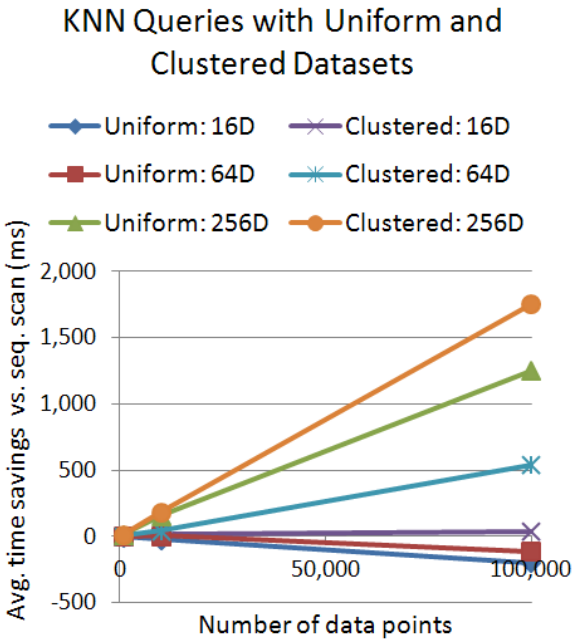

Fig. 9. Average time savings of iMinMax over sequential scan for KNN query ($k = 3$) with uniform and clustered datasets

well with the number of dimensions and the size of the data. Whereas inserting points into the tree does not depend on dimensionality, we see that the index build time increases for higher dimensions. This is because the iMinMax index value is determined by the "closest edge," which requires iterating through every dimension for every point.

We see similar point query and range query performance between the uniform, clustered, and TRACE datasets. The improvement over sequential scan becomes more predominant as the number of dimensions and size of the data increases. For smaller datasets in fewer dimensions, the extra overhead of maintaining and searching through the B$^+$-tree is not balanced by substantial performance gains, and the iMinMax algorithm performs on par with sequential scan.

We see in Fig. 9 that iMinMax performs *worse* than sequential scan *only* for KNN queries with low dimensionality but large data size.

VIII. Extension to Find Optimal θ

In the original iMinMax paper, setting the θ parameter to account for skewed distributions entailed stepping through several values of θ and evaluating the performance. In this extension, we apply the algorithm in [1] for calculating the approximate median and use this value to estimate an optimal θ.

A d-dimensional median point is used calculate the optimal θ, denoted θ_{opt}. This d-dimensional median is calculated by first finding the median for each dimension. The combination of these d one-dimensional medians forms the median point used to calculate θ_{opt}. The x_{min} and x_{max} of this median are then used to calculate the optimal θ as

$$\theta_{opt} = 1 - x_{max} - x_{min}, \tag{3}$$

which is derived from the conditional statement of the first case of equation (1). This θ_{opt} is intended to more evenly divide the data points between the Max and Min edges, which, in turn, results in more efficient indexing [6].

To test the performance of iMinMax using θ_{opt} we generated a dataset of 100,000 2D points with a roughly normal distribution centered at the point $(0.2, 0.2)$. We then calculated θ_{opt} as 0.6 using equation (3). Testing range queries above, below, and intersected by the split in the data generated by θ_{opt}, we compared the performance with trees generated by varying θ for both wide and narrow range queries.

Our results showed that the trees generated using θ_{opt} generally performed better. The θ_{opt} trees performed particularly

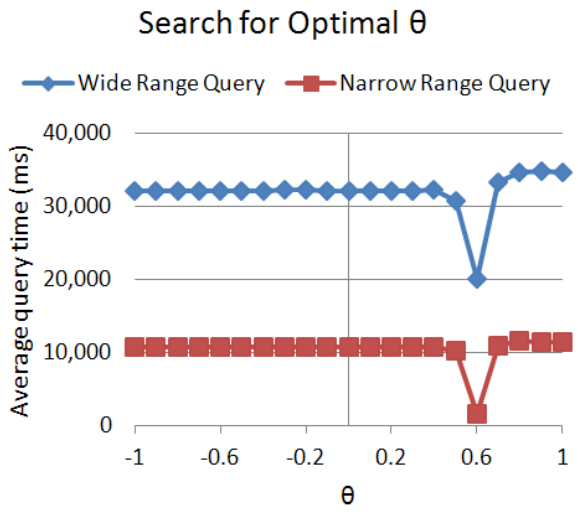

Fig. 11. Searching for optimal θ by brute force

well on queries that crossed the center of the distribution and on queries that were closer to center. These results confirm that finding θ_{opt} allows for more efficient indexing and that θ_{opt} can be found by calculating the median rather than having to perform a brute force search over different values of θ.

IX. Conclusion

We have demonstrated that the iMinMax algorithm generally offers robust improvements over sequential scan as the number of dimensions and size of the data increases, thereby reducing the effects of the "curse of dimensionality" problem.

Furthermore, we have shown that an optimal θ can be estimated algorithmically rather than having to iteratively search for it empirically. With a way to algorithmically fine-tune θ, the possibility of automatically calculating θ_i for each dimension individually, as proposed in [6] but never implemented, now becomes feasible.

Appendix

This project is hosted, with full source code, on Google Code at:

http://code.google.com/p/iminmax/.

An archive of the datasets used can be found at:

http://code.google.com/p/iminmax/downloads/detail?name=data.tar.bz2

The original raw experimental results are available at:

http://code.google.com/p/iminmax/source/browse/#git/main_program/results.

Finally, additional programming credits are noted at:

http://code.google.com/p/iminmax/source/browse/main_program/readme/credits.txt.

References

[1] Sebastiano Battiato, Domenico Cantone, Dario Catalano, Gianluca Cincotti, and Micha Hofri. An efficient algorithm for the approximate median selection problem. In Giancarlo Bongiovanni, Rossella Petreschi, and Giorgio Gambosi, editors, *Algorithms and Complexity*, volume 1767 of *Lecture Notes in Computer Science*, pages 226–238. Springer Berlin / Heidelberg, 2000.

[2] Stefan Berchtold, Christian Böhm, and Hans-Peter Kriegal. The pyramid-technique: towards breaking the curse of dimensionality. *SIGMOD Rec.*, 27:142–153, June 1998.

[3] Stefan Berchtold and Daniel A. Keim. High-dimensional index structures database support for next decade's applications. In *Proceedings of the 1998 ACM SIGMOD international conference on Management of data*, SIGMOD '98, pages 501–, New York, NY, USA, 1998. ACM.

[4] Christian Böhm. A cost model for query processing in high dimensional data spaces. *ACM Trans. Database Syst.*, 25:129–178, June 2000.

[5] H. V. Jagadish, Beng C. Ooi, Kian L. Tan, Cui Yu, and Rui Zhang. iDistance: An adaptive B^+-tree based indexing method for nearest neighbor search. *ACM Trans. Database Syst.*, 30(2):364–397, June 2005.

[6] Beng Chin Ooi, Kian-Lee Tan, Cui Yu, and Stephane Bressan. Indexing the edges a simple and yet efficient approach to high-dimensional indexing. In *Proceedings of the nineteenth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, PODS '00, pages 166–174, New York, NY, USA, 2000. ACM.

[7] Q. Shi and B. Nickerson. Decreasing Radius K-Nearest Neighbor Search Using Mapping-based Indexing Schemes. Technical report, University of New Brunswick, 2006.

[8] Shuguang Wang, Cui Yu, and Beng Ooi. Compressing the index - a simple and yet efficient approximation approach to high-dimensional indexing. In X. Wang, Ge Yu, and Hongjun Lu, editors, *Advances in Web-Age Information Management*, volume 2118 of *Lecture Notes in Computer Science*, pages 291–302. Springer Berlin / Heidelberg, 2001.

[9] Cui Yu, Stéphane Bressan, Beng Chin Ooi, and Kian-Lee Tan. Querying high-dimensional data in single-dimensional space. *The VLDB Journal*, 13:105–119, May 2004.

[10] Cui Yu, Beng C. Ooi, Kian-Lee Tan, and H. V. Jagadish. Indexing the Distance: An Efficient Method to KNN Processing. In *VLDB '01: Proceedings of the 27th International Conference on Very Large Data Bases*, pages 421–430, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc.