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.**

I. Introduction

Many efficient indexing structures exist for storing and querying low-dimensional data, such as found in spatial applications, which typically need only 2 or 3 dimensions to represent objects in space. When viewing the data space over a set of features rather than spatial coordinates, the number of dimensions will usually substantially increase. Furthermore, the queries used by spatial applications are still relevant in the high-dimensional feature space, such as point, range, and nearest-neighbor (similarity) queries.

However, many of these traditional indexing techniques efficient for low-dimensional data deteriorate as the number of dimensions increases, often becoming more expensive than sequential scan. To address this problem, several indexing techniques specifically meant for high-dimensional data have been developed.

II. Related Work

Several of the recent approaches for indexing high-dimensional data map each data point to a one-dimensional index (real value) [9]. The benefit of mapping to a single, real-valued index is being able to reuse the data structures, such as B$^+$-trees, already available in commercial DMBS software.

One such approach, called the Pyramid-Technique [1], 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. Thus, 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 actual 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.

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.

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

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.

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.

On the other hand, the KNN extension for iMinMax presented in [7] uses a decreasing radius technique. 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 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.

The iMinMax(θ) algorithms for point, range, and KNN queries were implemented in C++. We used the STX B$^+$-tree [2] in our implementation and used the getoptpp command line parser [4] for scripting the experiments. 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.

IV. Datasets

As [3] 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. Results

The results will go here.

A. Uniform

B. Clustered

C. Real-Life

VI. Discussion

Some discussion will go here.

VII. Conclusion

The conclusion will go here.

References

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

[2] Timo Bingmann. STX B+ Tree C++ Template Classes, November 2011.

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

[4] Daniel Gutson and Hugo Arregui. getoptpp - Yet Another getopt C++ version, STL-streaming like, November 2011.

[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.