CSCI 540 - Advanced Databases Course Project Progress Report for November 30th

Larry Lynn
Liessman Sturlaugson
Cole Schock

November 29, 2011

Abstract

The iMinMax team has implemented the algorithms for performing point, range, and k-nearest neighbor queries and have begun gathering the required measurements for B^+-tree statistics. The team has made progress on writing the background information for the final paper and have identified a list of coding tasks to complete before the team transitions into performing the full set of query experiments.

1 Code Progress

The team has implemented the original versions for the three required queries: point, range, and KNN. As mentioned in the future work of the original iMinMax paper [3], ideas to improve the algorithm involve automatically tuning the θ parameter and multiple θ_i. These would not significantly alter the original algorithms but serve as additions. In any case, the original algorithms serve as a baseline against which to compare any extensions which the team decides to pursue.

The algorithms listed below underwent initial testing using a small, 2-dimensional dataset for easy results verification, although code testing and clean-up are still ongoing.

1.1 Point Query

The point query algorithm, as with all the iMinMax query algorithms, follows a filter and refine strategy. Thus, each index value in the tree that matches that of the query point must be checked exactly. Furthermore, there may be multiple points that exactly match the query point. Thus the algorithm was written to return the set of all points that match.

1.2 Range Query

The range query algorithm accepts two points, representing opposite corners of a d-dimensional cube. The algorithm converts this hypercube range into d one-dimensional queries. The algorithm also checks if some of these subqueries may be pruned according to Theorem 2 of [3]. The algorithm filters and refines at each subquery, returning the set points falling into the range once all the subqueries have been evaluated.

1.3 KNN Query

The KNN query algorithm uses the decreasing radius strategy as described in [4]. The algorithm also makes use of the one-dimensional subquery search method used by the range query algorithm. At each range subquery, the algorithm maintains a list sorted by the distance from the query point. Thus, as closer points are found, they replace the current farthest points in the candidate set. Once all the subqueries have been evaluated, the algorithm returns the list sorted by nearest neighbor, along with each neighbor point's distance to the query point, as per the project requirements.

1.4 Exception Handling

The team also set up the framework for exception handling within the main() method. Exceptions thrown within the iMinMax class methods are passed to main() with a unique integer ID for displaying a descriptive error message. Exceptions are thrown when the program detects malformed points (e.g. points used as input for point, range, or KNN queries that do not match the dimensionality of the data or CSV files containing points of inconsistent dimensionality).

2 Tree and Index Creation Experiments

Having the algorithm for iMinMax(θ) index creation and the B^+-tree implementation [1] working, the team began initial experiments on tree and index constructions times and tree statistics with the 27 datasets provided. In particular, the project requirements listed tree size (number of levels and nodes) and construction times for the indices and the tree, whereas the other performance measures are specific to queries. The timing of index and tree construction was implemented with code adapted from [2]. The results are shown in Table 1 using a max leaf node size of 256 bytes.

These results show reasonable consistency across the datasets. The tree build time, the number of levels, and the number of nodes does not seem to be largely affected by increasing the dimensionality, which makes sense because iMinMax maps to a single dimension regardless of the dimensionality of the input data. On the other hand, the tree build time shows the effect of increased number of points, as insertion into the tree becomes slightly more costly as the tree grows. As well, the index build time shows the effect of increased dimensionality, as

Table 1: Index and B^+-tree Statistics

Dataset Name	Tree Build Time (s)	Index Build Time (s)	Tree Levels	Number of Nodes
16Dim10cluster1000	0.001180	0.003429	3	99
16Dim10cluster10000	0.014687	0.038553	4	999
16Dim10cluster100000	0.174178	0.400126	5	9998
64Dim10cluster1000	0.001215	0.006954	3	99
64Dim10cluster10000	0.015238	0.071550	4	999
64Dim10cluster100000	0.178797	0.774095	5	9998
256Dim10cluster1000	0.001173	0.020172	3	99
256Dim10cluster10000	0.014561	0.203053	4	999
256Dim10cluster100000	0.171380	2.058770	5	9998
uniform_16_1000	0.001196	0.003445	3	99
uniform_16_10000	0.014677	0.038235	4	999
uniform_16_100000	0.172558	0.396877	5	9998
uniform_64_1000	0.001238	0.006988	3	99
uniform_64_10000	0.015131	0.071977	4	999
uniform_64_100000	0.177883	0.749190	5	9998
uniform_256_1000	0.001233	0.019900	3	99
uniform_256_10000	0.014980	0.199845	4	999
uniform_256_100000	0.175725	2.068880	5	9998
Real-16-160-random	0.000160	0.000542	2	15
Real-16-1600-random	0.001930	0.005586	3	159
Real-16-16000-random	0.024359	0.061313	4	1598
Real-64-160-random	0.000162	0.001077	2	15
Real-64-1600-random	0.001989	0.011408	3	159
Real-64-16000-random	0.025811	0.118236	4	1598
Real-256-160-random	0.000163	0.003210	2	15
Real-256-1600-random	0.001984	0.031989	3	159
Real-256-16000-random	0.024821	0.327106	4	1598

iMinMax requires that the maximum and minimum value be found across all dimensions for each point, which incurs slightly greater cost as the number of dimensions increases.

3 Progress on the Final Paper

After setting up the IEEE conference LaTeX template, the team has sketched a rough outline of the paper and begun to write the initial sections, such as the introduction, related work, and the description of the iMinMax algorithm [3] with the KNN extension [4]. Experimental design and results will be incorporated as they are performed.

4 Outline of Remaining Coding Tasks

The team has identified the following outstanding work for completing the C++ code:

- Tree segments output
- Save/load the points data structure along with the filled B^+-tree
- Extra command line parameters for filename for save/load the filled B^+-tree and the points data structure
- Query parser to load in point, range, and KNN query files
- Switch and loop for executing parsed queries
- Query timer (include calculations for 'average search time')
- Output formatting for user-friendly graph generation
- Sequential search for testing against iMinMax
- Preprocessor for finding distribution median (heuristic or exact)
- Code to approximate optimal θ given median(s)
- Support separate θ_i for each dimension

Note that several of these later items are in addition to the base requirements, and therefore the query experiments can commence before these items are fully implemented.

References

[1] Timo Bingmann. STX B+ Tree C++ Template Classes, November 2011, https://idlebox.net/2007/stx-btree/.

[2] Dennis Muhlestein. Timing C/C++ Code on Linux, November 2011, http://allmybrain.com/2008/06/10/timing-cc-code-on-linux/.

[3] Beng Chin Ooi, Kian-Lee Tan, Cui Yu, and Stephane Bressan. Indexing the edgesa 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.

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