Data node splitting policies for improved range query efficiency in k-dimensional point data indexes

Evangelos Outsios, Georgios Evangelidis
Department of Applied Informatics
University of Macedonia
Thessaloniki, Greece
Email: {outsios, gevan}@uom.gr

***Abstract*—High dimensional vectors (points) are very common in image and video classification, time series data mining, and many modern data mining applications. One of the most popular classification methods on such data is k-Nearest Neighbor (kNN) searching. Unfortunately, all proposed and state-of-the-art multi-attribute indexes fall short in terms of usability as dimensionality increases. This is attributed to the "dimensionality curse" problem, according to which, range searching above 10 dimensions is as efficient as a sequential scan of the entire database. Thus, kNN searching, as a special case of range searching, has to benefit a lot if we find ways to increase the performance of indexes in high dimensions. In this paper, we deal with space partitioning indexes and we propose six data node splitting techniques. We examine their performance in terms of data node storage utilization and quality of space partitioning. These two conflicting goals are both essential for good range query performance. Our experiments with uniform and skewed data demonstrate that certain splitting techniques can perform satisfactorily.**

***Keywords*-multi-attribute point data indexes; average storage utilization; space partitioning quality; range query performance**

I. Introduction

The increased interest in access methods for multidimensional points or vectors (point access methods - PAMs) is attributed to the fact that data mining applications that deal with image, video or time-series data need to manipulate and analyze vast quantities of multi-dimensional vectors. The problem of "dimensionality curse" states that above 10 dimensions and depending on the dataset at hand, exhaustive search of the dataset is faster than using a PAM for the purpose of range or kNN queries [1], [2].

To make things worse, vectors in data mining applications can have hundreds of dimensions. Various dimensionality reduction techniques can be used to reduce their dimensionality down to 6 to 20 dimensions without losing much information. A viable choice for indexing such large point datasets are PAMs that index the multidimensional space without creating overlapping subspaces and perform well in low to medium dimensions. On the other hand, SAMs (spatial access methods) like the R-tree [3], index multidimensional points and objects with spatial extent and allow for overlapping subspaces.

In this paper, we assume that our data records are multi-dimensional points and that the index of choice is a PAM, e.g., the hB-tree [4], [5] or the KDB-tree [6], etc., that stores points in the leaf nodes and splits them in non-overlapping regions. We choose to split data nodes using hyperplanes, i.e., a single attribute, since this approach requires the smallest index term to describe the split. Only the splitting attribute and its value are required to be posted at the index level above (the parent of the overfull node) to describe the split. We experiment with various splitting techniques and report on their performance in terms of the average data node storage utilization and quality of space partitioning.

In Section 2, we discuss the problem of data node splitting and how it can affect the performance of an index structure. In Section 3, we present six data node splitting techniques, and, in Section 4, we describe the experiments we conducted on the performance of the splitting techniques. We conclude the paper in Section 5.

II. Data node splitting

Data nodes contain the records (multi-dimensional points) of the database. Each data node corresponds to a different page on disk. In a way analogous to the B+tree [7], when a data node becomes overfull, because of new point insertions, it has to be split. After the split we have two data nodes, the initial one occupying the same disk page and a new one occupying a new disk page. This process is repeated each time a data node becomes overfull.

Since the data node level naturally occupies the majority of nodes in a multi-dimensional index tree, it is crucial that an index structure achieves the best possible average storage utilization and space partitioning regardless of data distribution and order of data insertion.

In previous work [8], we discussed the criteria by which we decide to split a data node. In general, there are two splitting approaches to consider, namely, space-oriented or point-oriented splittings. We split a data node either as evenly as possible in terms of the data points it contains to maximize storage utilization, or as evenly as possible in terms of the space it occupies to achieve the best possible space partitioning.

The drawback of the first approach is that, since there is no guarantee on good space partitioning, the shape of the resulting data nodes may be irregular and range query searching may become inefficient. The drawback of the second approach is that there is no guarantee on data node storage utilization. Hence, any data node splitting technique should take into consideration these matters and try to achieve a trade-off between data node storage utilization and efficient range query searching.

In the following sections, we propose six splitting techniques and show the results from the experiments we carried out.

III. New data node splitting techniques

We present six different data node splitting techniques. They are based on two orthogonal policies: (a) the way the splitting attribute is chosen (best, random, round robin), and, (b) the way the splitting value for the chosen attribute is chosen (even point split, even space split). All possible combinations of the two policies give us the six different splitting techniques.

A. Best attribute for even point split

This technique first favors even point and then even space data node splittings. The algorithm that describes the sequence of the splitting criteria is as follows:

1: {**TECHNIQUE t1**}
2: **for** each attribute **do**
3: find the value that achieves the best possible point split (i.e., equal or as close as possible to a 1:1 split)
4: **end for**
5: Let A be the set of (attribute, value) pairs that achieve the best point split
6: **if** A contains just one pair P **then**
7: **return** P
8: **end if**
9: Let $B \subseteq A$ the set of pairs that split the longest side
10: **if** B contains just one pair P **then**
11: **return** P
12: **end if**
13: Let $C \subseteq B$ the set of pairs that achieve the most even space split
14: **if** C contains just one pair P **then**
15: **return** P
16: **end if**
17: **return** a random pair from C

The idea is that when we have more than one candidate attributes, we proceed and choose the attribute that splits the space as evenly as possible, by considering the attribute that avoids creating hyper-rectangles with long edges (lines 9 and 13).

B. Best attribute for even space split

This technique first favors even space and then even point data node splittings. The algorithm that describes the sequence of the splitting criteria is given below:

1: {**TECHNIQUE t2**}
2: **for** each attribute **do**
3: find the value that splits the space in two and achieves a 1:2 or better point split and put the (attribute, value) pair in set A1
4: **if** there is no such value **then**
5: adjust the value accordingly to achieve the best possible point split and put the (attribute, value) pair in set A2
6: **end if**
7: **end for**
8: if A1 is empty A=A2 else A=A1
9: Let $B \subseteq A$ the set of pairs that split the longest side
10: **if** B contains just one pair P **then**
11: **return** P
12: **end if**
13: Let $C \subseteq B$ the set of pairs that achieve the best point split
14: **if** C contains just one pair P **then**
15: **return** P
16: **end if**
17: Let $D \subseteq C$ the set of pairs that split closer to the middle of the edge
18: **if** D contains just one pair P **then**
19: **return** P
20: **end if**
21: **return** a random pair from D

Here, the idea is to exploit the fact that when data nodes are split in a 1:2 ratio in the worst case, they achieve an average storage utilization of about 67%, which is very close to the average storage utilization achieved for 1:1 splittings (about 70%) [4].

Although this technique favors even space splittings, its first concern is data node storage utilization (lines 2 through 8). So, if there is no attribute that in addition to the even space split it also achieves a "good" point split (1:2 or better), we first choose the best possible point split and then look for the best space split (the case that A1 is empty and A is assigned to A2 instead).

C. Round robin attribute for even point split

This technique is quite simple. We keep in each data node's control block a next_splitting_attribute field that denotes the next attribute that should be used for splitting that node. When the data node is split, both this node and the newly created node get this field updated to denote the "next" attribute in a round-robin fashion.

1: {**TECHNIQUE t3**}
2: Choose as splitting attribute the attribute stored in the next_splitting_attribute field in the control block of the data node
3: Find the best point split value for this attribute
4: Set the next_splitting_attribute field to be the "next" attribute in a round-robin fashion
5: **return** the (attribute, value) pair

For uniformly distributed points, this splitting technique achieves good storage utilization since nodes are always split with the best possible ratio. Also, gives good space partitioning since data nodes are always split along the longest side. The performance could be very poor for skewed data. Another drawback of the technique is that every data node must store the bookkeeping information of the attribute that should be used for the following split.

D. Round robin attribute for even space split

This technique is simple and very similar to the previous one. Now, we favor even space splits, but as in Technique t2, our first concern is storage utilization.

1: {**TECHNIQUE t4**}
2: Choose as splitting attribute the attribute stored in the next_splitting_attribute field in the control block of the data node
3: Find the best space split value for this attribute
4: **if** this value does not achieve a 1:2 or better point split **then**
5: adjust the value until a 1:2 point split is achieved
6: **if** the above step fails **then**
7: adjust the value until the best possible point split is achieved
8: **end if**
9: **end if**
10: Set the next_splitting_attribute field to be the "next" attribute in a round-robin fashion
11: **return** the (attribute, value) pair

For uniformly distributed points, this splitting technique achieves acceptable space partitioning since data nodes are always split along the longest side and acceptable storage utilization. The performance could be very poor for skewed data. Again, a drawback of the technique is that every data node must store the bookkeeping information of the attribute that should be used for the following split.

E. Random attribute for even point split

This technique is very similar to Technique t3, but it chooses the splitting attribute in a random way. For uniformly distributed points, it achieves good storage utilization since nodes are always split as close to a 1:1 ratio as possible. On the other hand, there is no guarantee on the quality of space partitioning. The performance could be very poor for skewed data.

1: {**TECHNIQUE t5**}
2: Choose a random splitting attribute
3: Find the best point split value for this attribute
4: **return** the (attribute, value) pair

F. Random attribute for even space split

This technique is very similar to Technique t4, but it chooses the splitting attribute in a random way. For uniformly distributed points, it achieves acceptable space partitioning and storage utilization. The performance could be very poor for skewed data.

1: {**TECHNIQUE t6**}
2: Choose a random splitting attribute
3: Find the best space split value for this attribute
4: **if** this value does not achieve a 1:2 or better point split **then**
5: adjust the value until a 1:2 point split is achieved
6: **if** the above step fails **then**
7: adjust the value until the best possible point split is achieved
8: **end if**
9: **end if**
10: **return** the (attribute, value) pair

IV. Experimental Evaluation

The techniques are either point-oriented (i.e., they emphasize on the even distribution of points between split nodes) or space-oriented (i.e., they emphasize on the even space partitioning between split nodes). Thus, we need to measure average data node storage utilization and also figure out a way to measure how good is the space partitioning achieved by each technique. Obviously, an exhaustive evaluation of the performance of range queries across all dimensions can be a measure of the quality of the space partitioning.

We tested the six splitting techniques using uniform and highly skewed computer generated k-dimensional points with values in (0, 1). We varied the value of k to 2, 3, 4, 6, 10, 15 and 20 dimensions. We chose datasets of 100K points, index nodes of 15 kd-tree nodes and data nodes of 30 points (records). These parameters created relatively high trees (five levels).

To compare the techniques in terms of how good they partition points, we calculated the average data node storage utilization for each technique in each dimensionality.

To compare the techniques in terms of how good they partition space, we conducted an elaborate series of range query experiments and we report the average ratio of visited over total data pages to answer these queries.

For each dimensionality, we chose to use all possible combinations of 100 non-overlapping range queries with 1% selectivity each. Our approach for performing the 100 range queries needs to divide the total space into 100 equally sized sub-regions. For k=2, we get 100 square sub-regions when using a step of 0.1 for each attribute. However, when k=3,

we cannot divide the space into 100 equally sized cubes. So, we ran three experiments (runs), using each time the whole extent for one of the attributes and using a step of 0.1 for the other two, and averaging the results in the end. For k=3 there are $\binom{3}{2} = 3$ such scenarios. In each run the space was partitioned into 100 hyper-rectangles with size 1 x 0.1 x 0.1 (i.e., 1% of the volume).

Similarly, in k dimensions, we ran $\binom{k}{2} = \frac{(k-1)\times k}{2}$ experiments and we averaged the results. In each run, the size of the range query was 1% of the space, since they were hyper-rectangles where two sides had size 0.1 and the rest 1.

In addition to reporting the average ratio of visited over total data pages to answer these range queries over all runs, we also report the variance of the visited pages for all runs. The variance is a clear indicator of how uniformly the space is partitioned among all dimensions. A small value means that space partitioning is balanced among all dimensions, whereas a large value means that space partitioning is imbalanced.

In general, we expect:

- The techniques that favor even point splitting should achieve good storage utilization for both uniform and skewed data (t1, t3, t5)
- The techniques that favor even space splitting should achieve better space partitioning and hence perform better in the range query experiments (t2, t4, t6)
- Technique t1 should outperform techniques t3 and t5
- Technique t2 should outperform techniques t4 and t6

In Figures 1, 2, 3 we report the storage utilization, variance, and visited pages for uniform data, and in Figures 4, 5, 6 we report the same results for skewed data.

Our experiments showed that for uniform data:

- **Utilization** All techniques have good storage utilization. Since data is uniformly distributed, the observed variation is coincidental. The expected average utilization is about 70%.
- **Variance** Techniques t1 and t2 guarantee balanced space partitioning regardless of k, since t2 is designed to favor data nodes whose space is as close to a hypercube as possible, and t1 also tries to achieve good space partitioning. As expected, the two round robin techniques (t3 and t4) get worse as k increases, whereas the random techniques have the worst performance in high dimensions.
- **Average pages accessed** Techniques t1, t2, t3 and t4 perform similarly as k increases and outperform the random attribute ones by about 10% over all dimensions. The "dimensionality curse" problem is evident, but it appears that for range queries with 1% selectivity the performance is acceptable. Most kNN queries correspond to range queries with much lower selectivity.

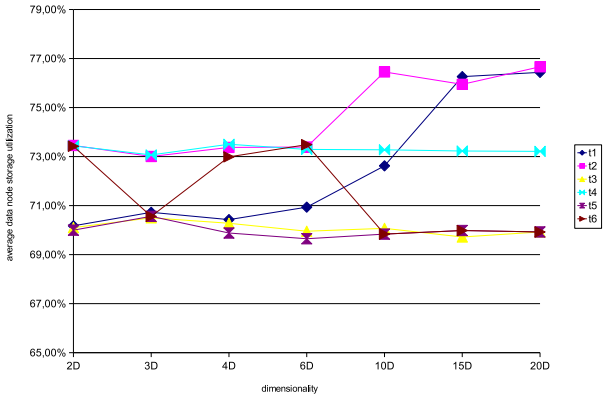

Figure 1. Data node storage utilization (uniform data)

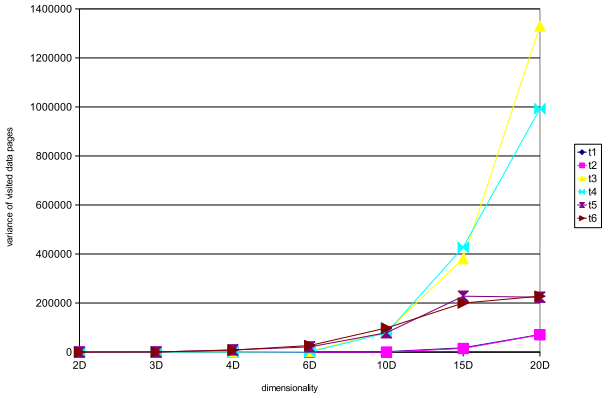

Figure 2. Variance of visited data pages per run (uniform data)

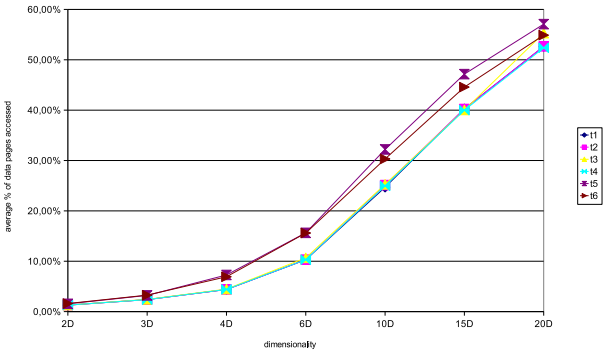

Figure 3. Average % of visited over total data pages (uniform data)

Similarly, for skewed data:

- **Utilization** Techniques t1, t3 and t5 (point-oriented) have good storage utilization as k increases. Techniques t2, t4 and t6 (space-oriented) perform worse since they favor even space partitioning.
- **Variance** Technique t2 outperforms all others since it favors balanced space partitioning. Technique t1 (point-

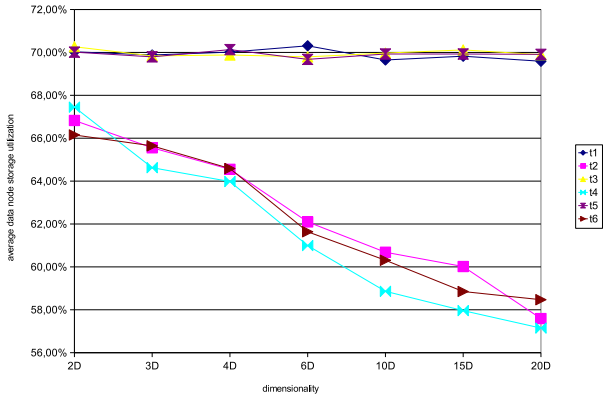

Figure 4. Data node storage utilization (skewed data)

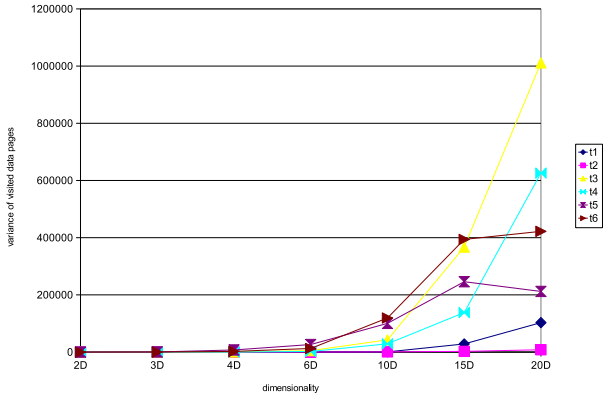

Figure 5. Variance of visited data pages per run (skewed data)

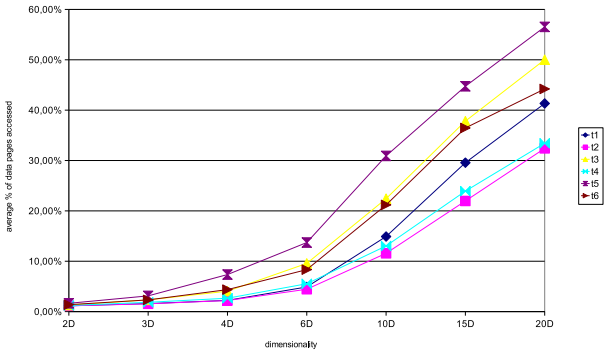

Figure 6. Average % of visited over total data pages (skewed data)

oriented) performs well too, since it takes into account good space partitioning, too.

- **Average pages accessed** As expected, Technique t2 has the best performance. Technique t4 performs very well, too. Again, the "dimensionality curse" problem is evident, although the performance is much better on skewed data. This is the strong point of point access methods over space access methods.

V. Conclusions

We presented six data node splitting techniques for point access methods that split space in non-overlapping regions. The techniques differ in the way they split overfull data nodes. They choose the splitting attribute and its value in order to achieve simultaneously even point and space splits. Since this is impossible to achieve unless data is uniformly distributed in space, we are interested in the performance of the various techniques when data is highly skewed.

Our experiments demonstrate that it is possible to achieve a trade-off between point and space partitioning and obtain even splittings over points and space at the same time. Two of the techniques we propose (t1 and t2) achieve both good space partitioning and good data node storage utilization.

References

[1] C. C. Aggarwal, "Re-designing distance functions and distance-based applications for high dimensional data," *SIGMOD Rec.*, vol. 30, pp. 13–18, March 2001. [Online]. Available: http://doi.acm.org/10.1145/373626.373638

[2] K. S. Beyer, J. Goldstein, R. Ramakrishnan, and U. Shaft, "When is 'nearest neighbor' meaningful?" in *Proceedings of the 7th International Conference on Database Theory*, ser. ICDT '99. London, UK: Springer-Verlag, 1999, pp. 217–235. [Online]. Available: http://portal.acm.org/citation.cfm?id=645503.656271

[3] A. Guttman, "R-trees: a dynamic index structure for spatial searching," in *Proceedings of the 1984 ACM SIGMOD international conference on Management of data*, ser. SIGMOD '84. New York, NY, USA: ACM, 1984, pp. 47–57. [Online]. Available: http://doi.acm.org/10.1145/602259.602266

[4] D. B. Lomet and B. Salzberg, "The hB-tree: a multiattribute indexing method with good guaranteed performance," *ACM Trans. Database Syst.*, vol. 15, pp. 625–658, December 1990. [Online]. Available: http://doi.acm.org/10.1145/99935.99949

[5] G. Evangelidis, D. Lomet, and B. Salzberg, "The hB^{Π}-tree: a multi-attribute index supporting concurrency, recovery and node consolidation," *The VLDB Journal*, vol. 6, pp. 1–25, February 1997. [Online]. Available: http://dx.doi.org/10.1007/s007780050030

[6] J. T. Robinson, "The K-D-B-tree: a search structure for large multidimensional dynamic indexes," in *Proceedings of the 1981 ACM SIGMOD international conference on Management of data*, ser. SIGMOD '81. New York, NY, USA: ACM, 1981, pp. 10–18. [Online]. Available: http://doi.acm.org/10.1145/582318.582321

[7] D. Comer, "Ubiquitous B-Tree," *ACM Comput. Surv.*, vol. 11, pp. 121–137, June 1979. [Online]. Available: http://doi.acm.org/10.1145/356770.356776

[8] E. Outsios and G. Evangelidis, "Achieving optimal average data node storage utilization in k-dimensional point data indexes," in *Proceedings of the 5th International Scientific Conference, eRA: The Contribution of Information Technology to Science, Economy, Society and Education*, 2010.