The BUB-Tree

Robert Fenk <fenk@forwiss.de>
FORWISS – Bavarian Research Center for Knowledge Based Systems
Orleansstrae 34, 81667 Munich, Germany

Abstract

Real world data has usually a non-uniform data distribution, i.e., there are clusters of data, but most of the universe is unpopulated space, the so called dead space. When indexing such data it is important to handle also dead space efficiently, i.e., the index should not degenerate with respect to size and performance when dealing with such non-uniformly distributed data.

The UB-Tree partitions the whole universe, i.e., it indexes the whole universe. For this reason, queries on dead space might perform badly, since they have to go down to the data page level in order to ensure there is no data answering the query.

To solve this problem of the UB-Tree, we propose the bounding UB-Tree (BUB-Tree), an UB-Tree storing additional information in the index part of the tree and modified algorithms utilizing this information. Query processing can utilize this information to prune search paths and consequently reduce the number of necessary data page accesses.

Keywords: *multidimensional point data, dead space, UB-Tree, range query*

1 Introduction

The UB-Tree partitions the whole universe into a set of disjunctive but consecutive Z-intervals, i.e., its index covers the whole universe. Queries on populated areas are handled well by this technique, but queries covering dead space might perform badly, since they have to traverse the tree down to the data page level in order to ensure there is no answer for the query.

To solve this problem of the UB-Tree we propose the bounding UB-Tree (BUB-Tree). Instead of storing just separators of Z-regions in the index part of the B-Tree it stores Z-intervals bounding the data stored on the corresponding page. Queries now can utilize this information during search to prune paths already at the index level and consequently the number of data page accesses can be reduced while preserving the good properties of the UB-Tree.

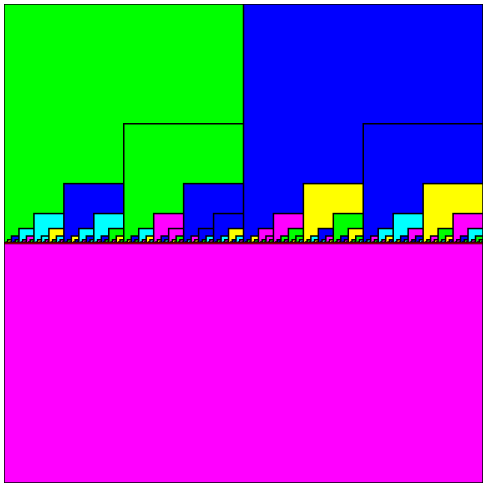
Related work are all papers on multidimensional access methods, especially the R-Tree [Gut84] and its variants the R^+-Tree and the R^*-Tree which are able to prune search paths to dead space in queries.

2 The UB-Tree

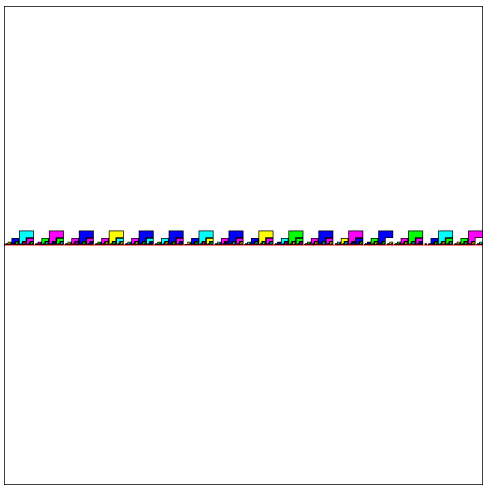
In this section we introduce the BUB-Tree an variant of the UB-Tree that can cope with dead space queries and non uniformly distributed data.

The UB-Tree [Bay97, Mar99] is a clustering index for multidimensional point data, which inherits all good properties of the B-Tree [BM72]. Logarithmic performance guarantees are given for the basic operations of insertion, deletion and point query, and a page utilization of 50% is guaranteed. The UB-Tree clusters data according to a space filling curve, namely the Z-curve [OM84], i.e., the multidimensional universe is linearized to a one dimensional space by representing a multidimensional point by its position on the Z-curve, the so called Z-address. The

UB-Tree introduces the new idea of partitioning the data space into disjoint Z-regions, which are mapped to disk pages. The Z-regions are then indexed by a B-Tree on the last included Z-address of a Z-region. These Z-regions in conjunction with a sophisticated algorithm for multidimensional range queries enable the UB-Tree to deal efficiently with multidimensional applications like data warehousing etc. Integrating the UB-Tree into a RDBMS providing a B-Tree is quite cheap as discussed in [RMF$^+$00].



(a) UB-Tree Z-regions



(b) BUB-Tree Z-regions

3 BUB-Tree Z-regions

A tuple in the index part of a (U)B$^+$-Tree consist of a separator and a link to the associates child node. Due to these separators the whole data space is partitioned into disjunctive Z-regions. Whether they cover dead space or not is not known at the index level.

So instead of storing a link and a Z-address which is a separator we store the Z-address of the first (*zfirst*) and last (*zlast*) tuple on the page, assuming tuples on a page are sorted with respect to the Z-order. Consequently index pages will contain fewer entries.

Fig. 1(a) shows the Z-region partitioning of the UB-Tree and Fig. 1(b) the partitioning of the BUB-Tree for a two dimensional universe where all the data is distributed at the bottom of the upper half of the universe. The UB-Tree has Z-regions covering large fractions of the universe but those Z-regions are only populated with data where they cover the the bottom of the upper half of the universe. In contrast to this the Z-regions of the BUB-Tree approximate just the area where data is located.

3.1 Point Query

Point queries are handled by traversing the index starting at the root page. We search for an index entry that contains the Z-address of the search point within its Z-interval. This can be done by binary search since the entries are sorted according to Z-order on the page. If we do not find such an entry, the point does not exist. If we find an entry, we follow the associated link to the next page and proceed as before as long as it is a index page. Otherwise we check if the point is stored on the data page. The worst case complexity of this search is the same as for a B-Tree, which is logarithmic to the size of the database.

3.2 Insertion, Split and Deletion

For insertion we do a point search, but instead of stopping if there is no matching index entry we traverse the path which is nearest to the given

point with respect to the Z-order. When reaching a data page we insert the point. A point inserted at the start or the end of the page triggers an update of the index entry leading to this page. This might propagate up to the root page.

Insertions that cause a page overflow are handled by splitting the page at the position of the biggest Z-difference (hole) between two consecutive page entries. Tuning parameters for this are the minimum fill rate, the size of the hole with respect to the Z-region volume resp. the universe volume. The dead space that is covered by the index can be minimized by decreasing the minimum fill rate, i.e., allowing a page utilization below 50%.

Deletion is handled by a point query and on success we delete the point from the data page. Deletion at the start or the end of the page triggers an update of the index entry leading to this page. This might propagate up to the root page. A page underflow is handled in the same way as for the B-Tree.

3.3 Range Query

Processing of a query box $[[ql, qh]]$ with $Z(ql) \leq Z(qh)$, where Z calculates the Z-address, starts with a point search for $Z(ql)$. If we find a data page we have to post-filter its content an continue with the next intersection (NI) of the Z-curve greater than *zlast*; otherwise we continue with the NI which is greater than $zstart$ of the last inspected entry. Processing is finished when reaching a Z-address eauql or greater than $Z(ql)$.

4 Preliminary Performance Results

Preliminary results are very promising. In our measurements the BUB-Tree index size was 3% bigger than the UB-Tree for optimal splits. For uniformly distributed data there was no difference with respect to data page accesses. For non-uniformly distributed artificial and real world data sets and queries the BUB-Tree was able to save 70% of the data page accesses in average. For some queries the BUB-Tree was loading no data pages while the UB-Tree was loading thousands of them.

5 Summary and Future Work

We have presented an enhanced variant of the UB-Tree, the so called BUB-Tree, which can cope with dead space queries.

In our future research we will investigate the performance gains in comparison to the R-Tree and its variants with artificial as well as with real world data sets.

References

[Bay97] Rudolf Bayer. The universal B-Tree for multidimensional Indexing: General Concepts. In *World-Wide Computing and its Applications '97 (WWCA '97), Lecture Notes on Computer Science*. Springer Verlag, 1997. Tsukuba, Japan.

[BM72] Rudolf Bayer and E. McCreight. Organization and Maintainance of large ordered Indexes. In *Acta Informatica 1*, pages 173–189, 1972.

[Gut84] Antonin Guttman. R-Trees: A Dynamic Index Structure for Spatial Searching. In Beatrice Yormark, editor, *SIGMOD'84, Proceedings of Annual Meeting, Boston, Massachusetts, June 18-21, 1984*, pages 47–57. ACM Press, 1984.

[Mar99] Volker Markl. *Processing Relational Queries using a Multidimensional Access Technique*. PhD thesis, DISDBIS, Band 59, Infix Verlag, 1999.

[OM84] Jack A. Orenstein and T. H. Merrett. A Class of Data Structures for Associative Searching. In *Proceedings of the Third ACM SIGACT-SIGMOD Symposium on PODS, April 2-4, 1984, Canada*, pages 181–190. ACM, 1984.

[RMF+00] Frank Ramsak, Volker Markl, Robert Fenk, Martin Zirkel, Klaus Elhard, and Rudolf Bayer. Integrating the UB-Tree into a Database System Kernel. In *Proc. of VLDB, 2000, Cairo, Egypt*, 2000.