Management and Query Processing of one dimensional Intervals with the UB-Tree *

Robert Fenk Volker Markl Rudolf Bayer

Bavarian Research Center for
Knowledge Based Systems
Orleansstraße 34, 81667 Munich, Germany
Phone: +49-89-48095-216, -191
Fax: +49-89-48095-203

{fenk,markl}@forwiss.de, bayer@in.tum.de

Abstract

The management and query processing of one dimensional intervals is a special case of extended object handling. One dimensional intervals play an important role in temporal databases and they can also be used for fuzzy matching, fuzzy logic and measuring quality classes, etc. Most existing multidimensional access methods for extended objects do not address this special problem and most of them are main memory access methods that do not support efficient access to secondary storage.

The research in the application of the UB-Tree to extended objects is part of my doctoral work. The contribution of this article is a specific solution for managing and querying one dimensional intervals with the UB-Tree, a multidimensional extension of the classical B-Tree. The combination of UB-Tree and transformation of extended objects to parameter space is an effective solution for this specific problem.

Keywords: one dimensional intervals, extended object handling, point query, range query, spatial data, parameter space

1 Introduction

The management of extended objects has been discussed a lot in the research community and several multidimensional access methods (MAM) have been proposed. R-Trees [8] and quad-trees [16] are a quasi standard in spatial database applications. These data structures have been developed to deal with extended objects with a dimensionality greater than one and they are not specialized for one dimensional objects resp. intervals. On the other hand there are specialized data structures like the external segment tree [5] which are rather complicated to integrate into a DBMS. However, the problem of efficient management and query processing of one dimensional intervals is an interesting question, because B-Trees and other common access methods of commercial RDBMS cannot handle them efficiently. From now on we call one dimensional intervals simply *intervals*.

In the following we will give examples of applications which require interval matching and management.

Temporal Databases: Standard databases store only the current version of the data – a kind of snap shot, but with temporal databases we have a history about all changes, versions and the lifetime of objects, which is seldom a point.

Fuzzy Logic/Matching: The application area of fuzzy logic resp. matching [9] also can be mapped to the one dimensional interval matching. Fuzzy matching is not a point query, but an interval match where the interval width determines the fuzziness of the match.

And also the following applications may be modelled by intervals.

Personalization: Personalizing [17] a web-page requires to track the user behavior and finding a given profile which fits this user best. The behavior of the user and typical profiles can be expressed as a set of intervals over all accessed files. For assigning a profile to a user, one has to match the user intervals with the profile intervals and find the best matching profile, which is the profile that best overlaps the user's intervals.

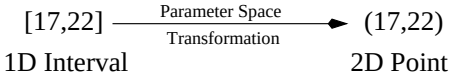


Figure 1: Endpoint Transformation

Quality Classes: Products like CPUs, cathode ray tubes and others are produced on the same assembly line and have to be sorted according to some quality criteria (e.g., clock rate of the CPU). Again the properties may be expressed by a set of intervals and finding the right class is interval matching.

Commercial RDBMSs usually offer none or only limited support for interval processing. The problem of most MAMs is that they are designed as main memory data structures and therefore do not support efficient retrieval from secondary storage. The drawback of others is the complexity of integrating them into existing relational DBMS technology.

We will show how intervals can be supported with only minor modifications to a RDBMS implementation containing a UB-Tree [2, 3] – with TransBase/Hypercube the first commercial RDBMS with UB-Tree support is available. The ideas are demonstrated in the context of point containment (find all intervals containing a specified point) and the intersection problem (find all intervals intersected by a specified interval).

2 Interval Handling with the UB-Tree

The UB-Tree, as well as other MAMs, cannot efficiently support interval handling by itself, but in combination with the transformation of intervals to parameter space it offers an efficient solution which is easy to integrate into a commercial RDBMS.

2.1 The Parameter Space

Simple geometric shapes can be considered as points in higher dimensional space called the *parameter space* [15]. The required transformation of an interval in one dimensional space interprets the beginning and end of the interval as coordinates in two dimensional space. This transformation is called *endpoint transformation* and results in a parameter space with two dimensions of equal domain.

Fig. 2(a) shows the intervals A, B, C and D, a point query PQ and an interval query IQ. The points in parameter space corresponding to the intervals are depicted in Fig. 2(b)-2(d). Points would map to the main diagonal, because they are intervals of width 0. Formulating queries is now

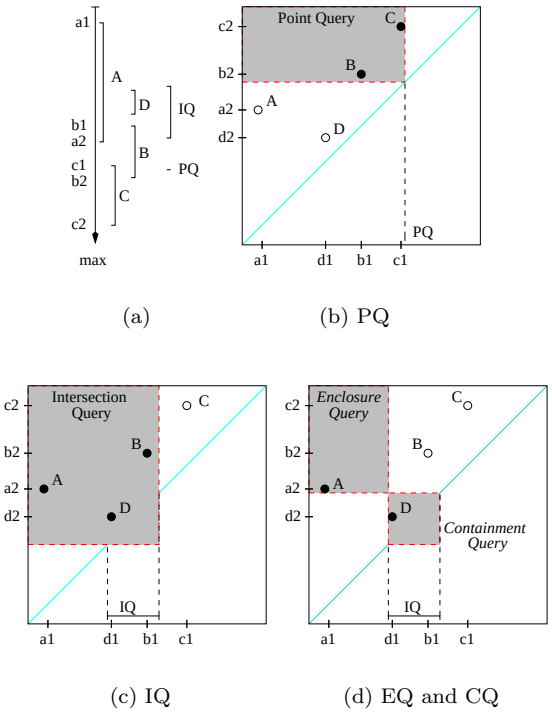


Figure 2: Native Space and Queries in Parameter Space

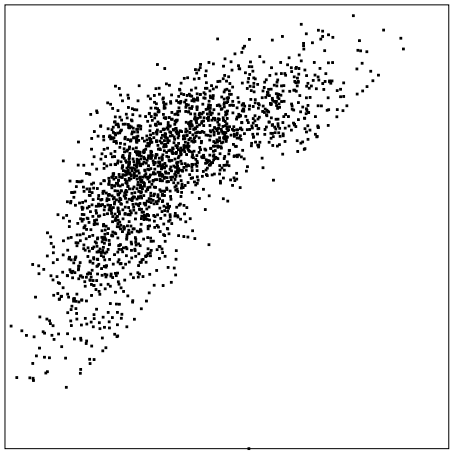
quite simple. As domain for the intervals we assume $U = [0, max]$. A point query (PQ) specified by $p \in U$ can be answered by the query box $[(0, max), (p, p)]$ (Fig. 2(b)). An intersection query (IQ) specified by $[s, e]$ (where $s, e \in U$) maps to the query box $[(0, max), (e, s)]$ (Fig. 2(c)). Similar, containment queries (CQ) map to the query box $[(0, max), (s, e)]$ and enclosure queries (EQ) map to the query box $[(s, e), (e, s)]$ (Fig. 2(d)).

All the necessary transformations are fairly simple and result in iso-oriented query boxes. However, another transformation method the *midpoint transformation* [7] has the problem that all typical queries result in query boxes, which are not iso-oriented. Therefore, we do not consider this transformation method.

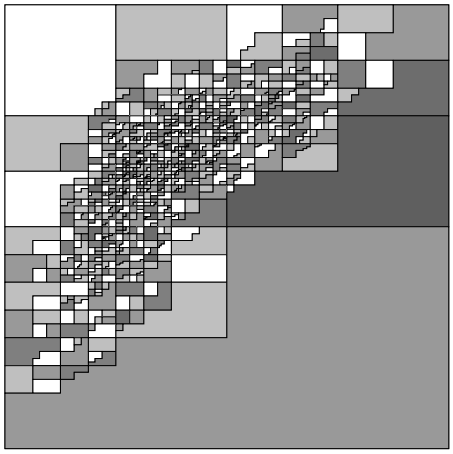
Despite the conceptual elegance, the parameter space approach has several disadvantages [7, 15], but they can be neglected in case of intervals, since then parameter space always has two dimensions.

2.2 Evaluation

Now we use the UB-Tree to index the parameter space. For example Fig. 3(a) shows two thousand Gaussian distributed intervals in parameter space and Fig. 3(b) the resulting UB-Tree partitioning of the parameter space. As one can see from the space partitioning, the UB-Tree adapts quite well to the



(a) Parameter Space



(b) UB-Tree

Figure 3: Distribution of intervals in parameter space and UB-Tree partitioning

non-uniform data distributed in parameter space. Earlier measurements [4, 11, 12, 6] have proven that its performance is excellent for up to six dimensions.

There is an constant overhead for the PQ, IQ and EQ, because the query box resulting from the parameter space transformation always starts in the upper left corner $(0, max)$. However, there are only a few UB-Tree regions covering a large sparsely populated area. Further they should remain in cache memory, because most of the queries require them.

To evaluate this approach further we check the requirements a MAM should meet according to [7]. Our approach inherits the following good properties of the UB-Tree [13]: excellent secondary storage management, logarithmic worst case guarantees for all major operations, a query response time that is linear to the size of the result set, independence of input data and insertion sequence, scalability to database growth and small space requirements for the index.

Both the UB-Tree and the necessary parameter space transformations are straight forward approaches and they are easy to implement and robust to be used in large-scale applications. Additionally they can be integrated with minimum impact to existing parts of a RDBMS, as long a it supports a B-Tree. Efficient support for interval management and querying would only require a new data type for intervals. Only minor changes to existing code of a RDBMS are necessary.

3 Summary and Outlook

We have presented a hybrid method to manage and query intervals efficiently. It transforms one dimensional intervals to two dimensional parameter space and indexes that space with a UB-Tree. The required transformations to parameter space are simple and independent of other RDBMS functionality.

In our further research we will investigate the usability and performance of the described approach with artificial data and, if possible, with real world data. In order to check the scalability and secondary storage management we will use databases which do not fit into main memory (e.g., several gigabytes of data). Otherwise all data could be cached in main memory and the tested access method has to be rather considered as a main memory access method and not a secondary storage access method. We will also check how other space filling curves instead of the UB-Tree's Z–curve result in better performance.

Finally we may investigate how this approach applies to extended objects in higher dimensional space.

References

[1] Bayer, R. and E. McCreight (1972); Organization and Maintainance of large ordered Indexes. Acta Informatica 1 (pp. 173-189)

[2] Bayer, R. (1996); The universal B-Tree for multidimensional Indexing. Technical Report TUM-I9637, Institut für Informatik, TU München

[3] Bayer, R. (1997); The universal B-Tree for multidimensional Indexing: General Concepts. World-Wide Computing and its Applications '97 (WWCA '97), Tsukuba, Japan, 10-11, Lecture Notes on Computer Science, Springer Verlag

[4] Bayer, R. (1998); R. Bayer and V. Markl. The UB-Tree: Performance of Multidimensional

Range Queries. Technical Report TUM-I9814, Institut für Informatik

[5] Blankenagel, G. and H. Güting (1994); External Segment Trees. Algorithmica 12(6) (pp. 498-532)

[6] Fenk, R., Markl, V. and R. Bayer (1999); Improving Multidimensional Range Queries of non rectangular Volumes specified by a Query Box Set. Proc. of International Symposium on Database, Web and Cooperative Systems (DWACOS), Baden-Baden, Germany

[7] Gaede, V. and O. Günter (1998); Multidimensional Access Methods. ACM Computing Survey 30(2) (pp. 170-231)

[8] Gutman, A. (1984); R-Trees: A dynamic index structure for spatial searching. Proc. of ACM SIGMOD Conf. (pp. 47–57)

[9] Kaufmann, M. (1991); Pearl, Judea: Probabilistic reasoning in intelligent systems : networks of plausible inference / Judea Pearl. Rev. 2. print. - San Mateo, Calif. ISBN 0-934613-73-7

[10] Kießling, W. and G. Köstler (1998); Multimedia-Kurs Datenbanksysteme. Berlin, Heidelberg. Springer Verlag, ISBN 3-540-63836-9

[11] Markl, V., Zirkel, M. and R. Bayer (1999); Processing Operations with Restrictions in Relational Database Management Systems without external Sorting. Proc. of ICDE Conf., Sydney, Australia

[12] Markl, V., Ramsak, F., and R. Bayer. (1999); Improving OLAP Performance by Multidimensional Hierarchical Clustering. Proc. of IDEAS Conf., Montreal, Canada

[13] Markl, V. (1999); MISTRAL: Processing Relational Queries using a Multidimensional Access Technique. DISDBIS, Band 59, Infix Verlag, ISBN 3-89601-459-5

[14] Orenstein, J.A. and T.H. Merret (1984); A Class of Data Structures for Associate Searching. Proc. of ACM SIGMOD–PODS Conf. (pp. 294–305)

[15] J. Orenstein (1990); A Comparison of Spatial Query Processing Techniques for Native and Parameter Space Proc. of ACM SIGMOD–PODS Conf. (pp. 343–352)

[16] Samet, H. (1984); The quadtree and related hierarchical data structures. ACM Computing Surveys 16(2). (pp. 187–260)

[17] Kahabka, T., Korkea-aho, M. and G. Specht (1997); GRAS: An Adaptive Personalization Scheme for Hypermedia Databases. Proc. of the 2nd. Conference on Hypertext - Information Retrieval - Multimedia (HIM '97), (pp. 279 - 292)