A Cost Function for Uniformly Partitioned UB-Trees

Volker Markl *Rudolf Bayer*

Bayerisches Forschungszentrum
für Wissensbasierte Systeme
Orleansstraße 34
81667 München
Germany

volker.markl@forwiss.de, bayer@in.tum.de

Abstract

Most operations of the relational algebra or SQL - like projection with duplicate elimination, join, ordering, group by and aggregations - are efficiently processed using a sorted stream of tuples. Often these operations are combined with restrictions in one or several attributes. Previous research has proposed algorithms for efficiently dealing with this kind of query pattern, which is highly relevant with respect to both data warehousing, data mining and GIS systems. In this paper we present a cost model that enables a concise estimation of both memory costs and run-time costs for processing queries with restrictions in multiple attributes that may in addition involve a sort operation. Our cost model considers uniformly distributed UB-Trees with independent dimensions and is derived analytically in three steps, starting with a very simple perfectly idealized partitioning scheme, moving on to imperfect partitioning schemes and finally evaluating a probabilistically partitioned UB-Tree. We also reason to what extent our cost model might be taken into account if the data is not distributed uniformly by investigating the connection between the cost model and selectivities.

1. Introduction

The UB-Tree is a multidimensional access method that relies on Z-ordering for multidimensional clustering. During an ESPRIT project we have integrated the UB-Tree [Bay97] into the kernel of the commercial relational DBMS TransBase [RMF+00]. The full integration of a multidimensional index into a DBMS not only speeds up range queries, it also ensures that the index is transparent to the user and solves concurrency and recovery issues. In contrast to all other relational DBMS, the UB-Tree in TransBase is not an add-on, but is fully integrated into the kernel. UB-Trees are almost transparent to the user, since queries are issued via standard SQL. Thus the user can define a multidimensional index on a set of attributes and use standard SQL for data manipulation and retrieval, and does not have to learn new constructs to especially work with multidimensional add-ons. For that reason, existing applications with standard SQL can take advantage of the performance benefits of multidimensional access methods.

If query optimization is to take a multidimensional index into account, a cost model for range queries is necessary. Since one additional benefit of a multidimensional index is the ability to use Tetris techniques for processing range queries with sort operations, we also present a cost analysis for this operation. Analyzing the cost of query processing with certain access methods is crucial to cost-based query optimization. Next to that it allows one to simulate query processing without actually creating the database and thus saves time and resources while gaining a better understanding of access methods. In addition a cost function is a benchmark for the query performance since it defines the expected response time and thus allows one to judge the quality of an access method implementation. Cost functions can further be used to predict the result sizes of a query or tell a user the expected processing time of a query before query execution has started. We define cost functions for page accesses for idealized uniformly partitioned universes, i.e., a multidimensional universe with uniformly distributed data in each dimension and independent dimensions. However, we also sketch how our approach is generalized to arbitrary independent data distributions by the use of histograms. We prove the quality of our cost function by comparing the predicted

number of page accesses with the actual number of page accesses measured with our prototype implementation of the UB-Tree. We also explain how our cost function is linked to the selectivity of a multidimensional query box.

The paper is organized as follows: Section 2 gives a brief survey on related work. In Section 3 the UB-Tree, the base access structure of our cost model is introduced. The cost model for uniformly partitioned UB-Trees is derived in Section 4 in an incremental way, starting with perfectly idealized partitioned UB-Trees and moving on to probabilistically partitioned UB-Trees. Section 5 gives our conclusions as well as an outlook on our future work.

2. Related Work

Much work has been published on the cost of access methods and sort operations for optimizing queries in single-database or multi-database applications by academia and has been applied by industry [ROH99, GGS96]. [Knu73] gives a comprehensive overview of searching and sorting as well as a careful and detailed cost analysis of related algorithms. [AV88] and [Vit99] focus on external memory algorithms. Much work about cost analysis has also been done for query algorithms in database systems. [HR96] analyzes the cost of join operations in this context. [Mer81] focuses on merge-sort operations. [Gra93] investigates a broad variety of query processing methods. [GG97] as well as [Gre89] investigate multidimensional access methods for multi-attribute searching. Many textbooks deal with cost models for classical DBMS algorithms, e.g., [Dat88], [GR97], [Ull88], [Ull89]. Many cost models have also been proposed focusing on special query types like nearest neighbor queries [BBK+97, CNP99] or access structures like R-Trees [TSS00, AS94] or Grid-Files [Bec93]. In contrast to our paper none of the previous work has addressed the cost of processing range queries with the UB-Tree as well as the cost of simultaneously handling range queries and sort operations by a single operator, the Tetris algorithm [MZB99]. Our work therefore extends previous work by giving new cost formulas for the UB-Tree and Tetris query processing methods. The contribution of our paper in combination with existing work enables to design a query optimizer or prediction unit for a database system, which in addition to traditional processing techniques also exploits the capabilities of UB-Tree and Tetris algorithms.

3. The UB-Tree

The basic idea of the UB-Tree [Bay97] is to use a space-filling curve to partition a multidimensional universe. Using the Z-Curve (Figure 1) the UB-Tree preserves the multidimensional clustering. A *Z-Address* α = Z(x) is the ordinal number of the key attributes of a tuple x on the Z-Curve, which can be efficiently computed by bit-interleaving (cf. Algorithm 1 or [OM84]). A standard B-Tree is used to index the tuples taking the linear Z-Address of the tuples as keys.

The fundamental innovation of UB-Trees is the concept of *Z-Regions* to create a disjunctive partitioning of the multidimensional space. This allows for very efficient processing of multidimensional range queries [Mar99]. A Z-Region [α : β] is the space covered by an interval on the Z-Curve and is defined by two Z-Addresses α and β. We call β the *region address* of [α : β].Each Z-Region maps exactly onto one page on secondary storage, i.e., to one leaf page of the B-Tree.

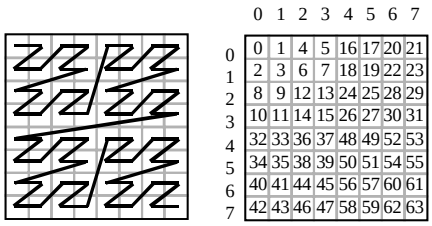

Figure 1: Z-ordering

For an 8×8 universe, i.e., s = 3 and d = 2, Figure 2 shows the corresponding Z-addresses. Figure 1c shows the Z-region [4: 20] and Figure 1d shows a partitioning with five Z-regions [0 : 3], [4 : 20], [21: 35], [36 : 47] and [48 : 63]. Assuming a page capacity of 2 points, Figure 1e shows ten points, which create the partitioning of Figure 1d. The details of the UB-Tree algorithms are described in [Bay97, Mar99].

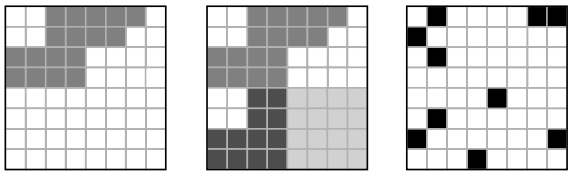

Figure 2: Z-regions

Definition 1: The *number of steps for attribute* A_i of a domain with cardinality r_i is determined by its resolution:

$$\text{steps}(i) = \log_2 r_i$$

Definition 2: The *length of step* k in bits (i.e., the number of dimensions in step k) is:

$$\text{steplength}(k) = |i \mid \text{steps}(i) \geq k \text{ and } i \in D\}|$$

```
Input:  x : tuple
Output: Z-address α

for step = 1 to max({steps(r_j) | j ∈ D})
   for i = 1 to steplength(step)
      copy bit step of x_i to bit i of α_step
   end for
end for
```

Algorithm 1: Bit-Interleaving

With $r = \max(\{\text{steps}(r_j) \mid j \in D\})$ bit interleaving has a CPU-complexity of $O(d \cdot r)$ bit operations The same holds for the inverse algorithm Z^{-1} that calculates the Cartesian coordinates of a tuple from its address.

4. The Cost of UB-Tree Range Queries

For a cost analysis of the UB-Tree range query performance it is necessary to have a cost function for the retrieved pages, i.e., the regions overlapped by a query box. This enables the prediction of the run time of a range query and yields a base for query optimization. In addition, a cost function permits to produce a statistical relevant number of measures by simulating range queries with varying table sizes and dimensionalities. This is especially useful, when practical measurements with our prototype implementation cannot be performed due to their memory requirements or their long run time. The cost function also allows a theoretical analysis of the range query performance and provides an excellent insight in the impact of the attribute order on a multidimensional index based on bit-interleaving.

To be useful for a broad range of applications, a cost function should not require too much knowledge about the queried database. The minimum requirements for input parameters of a cost function are the database size and the query restriction. For multi-dimensional index structures the dimensionality of the index is also necessary, i.e., the number of attributes contributing to the index. These parameters are in general easy to obtain and maintain.

To derive a cost function using only these input parameters is not achievable in general, since the data distribution is another decisive factor for the query performance. Yet it is possible to develop such a cost function for a certain case of space partitioning, the so-called idealized uniform partitioning, consisting of uniformly distributed and independent attributes.

4.1 A Cost-Function for Perfect Idealized Uniform Partitioning

In the following we use P for the number of data pages of a table. For a multidimensional partitioning we assume that the data has been partitioned over a total of d dimensions. A multidimensional interval (or query box) is specified by the lower bounds vector y and the upper bounds vector z. y_i and z_i denote the lower and upper limit of the restriction in dimension i. For two points (or tuples) y, z the multidimensional interval $[[y, z]]$ is denoted by:

$$[[y, z]] = [y_1, z_1] \times \ldots \times [y_d, z_d] =$$
$$= \{(x_1,\ldots,x_d) \mid y_i \le x_i \le z_i \text{ for all } i \in D\}.$$

The volume of a multidimensional interval $[[y, z]]$ is calculated by multiplying the lengths of each of the one-dimensional intervals $[y_i, z_i]$. Without loss of generality we assume the multidimensional space in each dimension to be normalized to the interval [0,1].

Definition 3 (perfect idealized uniform partitioning): A *perfect idealized uniform partitioning* splits the multidimensional space in P hypercubes (subcubes) with volume $2^{-d \cdot k}$, i.e., $P = 2^{d \cdot k}$ for some $k > 0$, $k \in \mathbb{N}$ (cf. to Figure 3 for a two-dimensional example). Otherwise we call an idealized uniform partitioning *imperfect*.

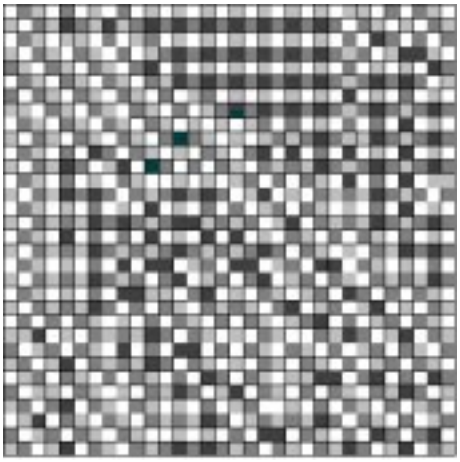

Figure 3: Perfect idealized uniform partitioning

Thus for perfect idealized uniform partitioning each dimension j of the universe has been partitioned by recursively dividing the universe $l_j(d, P) = \log_2 P / d = k$ times. We call $l_j(d, P)$ the number of split levels in dimension j. For perfect idealized uniform partitioning the number of Z-regions intersected by a query box [[y, z]] is identical to the number of subcubes overlapped by [[y, z]] (cf. Figure 4).

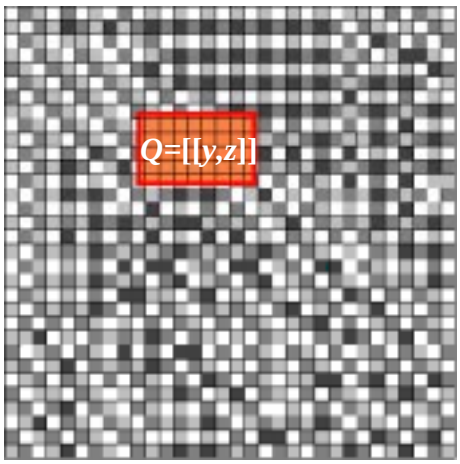

Figure 4: A Range Query on a Perfect idealized uniform partitioning

For an easy mathematical treatment we therefore define ^, a normalization operation of each domain to a

value of the interval [0, 1]. If a is a value in a domain $\mathbb{D} = [a_{\min}, a_{\max}] \subset \mathbb{N}_0$, then we normalize in the following way:

$$\hat{a} := \frac{a - a_{\min} + 1}{a_{\max} - a_{\min} + 1}$$

If we have l_j completed split levels in dimension j, the number of slices of the multi-dimensional space that are overlapped by the query box [[y, z]] in dimension j can be determined by calculating the number of the slices, that are contained in interval [0, $\hat{z}_j$], but not in interval [0, $\hat{y}_j$], i e., the number of slices in [0, $\hat{z}_j$] minus the number of slices in [0, $\hat{y}_j$]. Since the slice containing $\hat{y}_j$ is also a slice overlapped by the query box, we must increment the above number by one to get the correct number of slices. If the number of slices for the interval [0, $\hat{c}$] is calculated as $\lfloor \hat{c} \cdot 2^{l_j} \rfloor$, a non-existing slice $2^{l_j}+1$ is added for $\hat{z}_j = 1$ by the formula derived above. We must correct this error for the case $\hat{y}_j < 1$ and $\hat{z}_j = 1$. This is achieved by decrementing the number of slices by one. For $\hat{y}_j = \hat{z}_j = 1$ the subtraction removes the error, therefore no correction is necessary here.

Thus the number of slices $n(y_j, z_j, l_j)$ in dimension j overlapped by the query interval [y_j, z_j] for l_j completed splits in dimension j can be calculated by the following formula:

$$n(y_j, z_j, l_j) = \begin{cases} 2^{l_j} - \lfloor \hat{y}_j 2^{l_j} \rfloor & , \text{if } \hat{z}_j = 1 \wedge \hat{y}_j \neq 1 \\ \lfloor \hat{z}_j 2^{l_j} \rfloor - \lfloor \hat{y}_j 2^{l_j} \rfloor + 1 & , \text{otherwise} \end{cases}$$

The number of subcubes intersected by the query box $c(y, z, P, d)$ then is the product of $n(y_j, z_j, l_j(d, P))$ over all dimensions:

$$c(d, P, y, z) = \prod_{j=1}^{d} n\left(y_j, z_j, l_j(d, P)\right)$$

4.2 A Cost-Function for Semi-Perfect Idealized Uniform Partitioning

We call a partitioning an imperfect idealized uniform partitioning, if it produces rectangular regions, where each region has either the shape of a subspace with volume $2^{-\lceil \log_2 P \rceil}$ or consists of two of these subspaces. In this case the multidimensional space has been partitioned recursively $\lfloor \log_2 P \rfloor \bmod d$ times for some dimensions and $\lfloor \log_2 P \rfloor \bmod d + 1$ times for some other dimensions. One dimension may exist, where some parts of the space already have been partitioned $\lfloor \log_2 P \rfloor \bmod d + 1$ times, while other parts of the space only have been partitioned $\lfloor \log_2 P \rfloor \bmod d$ times.

Because of the above considerations we distinguish two cases of imperfect partitioning:

Definition 4 (semi-perfect and probabilistic idealized uniform partitioning): If $P = 2^k$ for some $k > 0$, $k \in \mathbb{N}$, we call an imperfect idealized uniform partitioning *semi-perfect*. Otherwise we call it *probabilistic*.

Definition 5 (probabilistic dimension): For a probabilistic idealized uniform partitioning we call a dimension *probabilistic*, if with respect to this dimension some parts of the space have been partitioned $\lfloor \log_2 P \rfloor \bmod d + 1$ times, while other parts of the space only have been partitioned $\lfloor \log_2 P \rfloor \bmod d$ times.

Lemma 1: Each probabilistic idealized uniform partitioning has exactly one probabilistic dimension.

Proof:

Bit interleaving takes place in a fixed order of dimensions. Our implementation of bit interleaving starts with the rightmost dimension. 2^l splits need to take place to completely split the space with respect to split level l (i.e., bit l of the binary representation of the Z-address). After these 2^l splits have taken place (i.e., enough data has been inserted into the UB-Tree), the next split takes place at split level $l + 1$ (i.e., bit $l + 1$ of the binary representation of the standard address). This split level corresponds to the next bit in the binary representation of standard addresses as obtained by bit interleaving. Therefore it splits the next dimension in the order of dimensions as used by bit interleaving.

Since splits complete one split level before moving to the next split level, only one dimension may have both subspaces with split level l and subspaces with split level $l + 1$. □

As a consequence of the proof of Lemma 1 the index of the probabilistic dimension in the order of dimension as used by bit interleaving [OM84] is calculated as:

$$\text{probabilistic}(d, P) = d - \left(\lfloor \log_2 P \rfloor \bmod d\right)$$

Example:

Split levels for perfect and imperfect uniform partitioning are illustrated in Figure 5 for a 6-dimensional space: For a table size of 64 pages the space is perfectly partitioned ($k = 1$, $d = 6$) with one split level for each dimension. With 512 ($k = 9$) pages this space is partitioned semi-perfectly with one split level in the first three dimensions and two split levels for the last three dimensions. With a page number of 700, the partitioning is probabilistic with dimension 3 as probabilistic dimension.

Pages	Split Levels per Dimension					
	Dim 1	*Dim 2*	*Dim 3*	*Dim 4*	*Dim 5*	*Dim 6*
64 (perfect)	1	1	1	1	1	1
512 (semi-perfect)	1	1	1	2	2	2
700 (probabilistic)	1	1	>1	2	2	2

Figure 5: Split levels

For a semi-perfect idealized uniform partitioning the number of completed splits $l_j(d,P)$ with respect to dimension j is calculated as

$$l_j(d,P) = \begin{cases} l_{j\downarrow}(d,P)+1 & \text{, if } \lfloor \log_2 P \rfloor \bmod d \le j \\ l_{j\downarrow}(d,P) & \text{, otherwise} \end{cases}$$

$$\text{where } l_{j\downarrow}(d,P) = \left\lfloor \frac{\log_2 P}{d} \right\rfloor$$

With $l_j(d,P)$ as defined above the $c(y, z, P, d)$ is calculated in the same way as for perfect idealized uniform partitioning:

$$n(y_j, z_j, l_j) = \begin{cases} 2^{l_j} - \lfloor \hat{y}_j 2^{l_j} \rfloor & \text{, if } \hat{z}_j = 1 \wedge \hat{y}_j \neq 1 \\ \lfloor \hat{z}_j 2^{l_j} \rfloor - \lfloor \hat{y}_j 2^{l_j} \rfloor + 1 & \text{, otherwise} \end{cases}$$

The key attributes are independent and the query box $[[x, y]]$ is iso-oriented with respect to each dimension. Therefore the total number of pages is obtained by multiplication of the slices in each dimension:

$$c(d,P,y,z) = \prod_{j=1}^{d} n\left(y_j, z_j, l_j(d,P)\right)$$

4.3 A Cost-Function for Probabilistic Idealized Uniform Partitioning

If an idealized uniform partitioning is probabilistic (cf. Figure 6 for an example), the formula $n(y_j, z_j, l_j(d, P))$ needs to be modified for the probabilistic dimension to take the probability of an incomplete split into account.

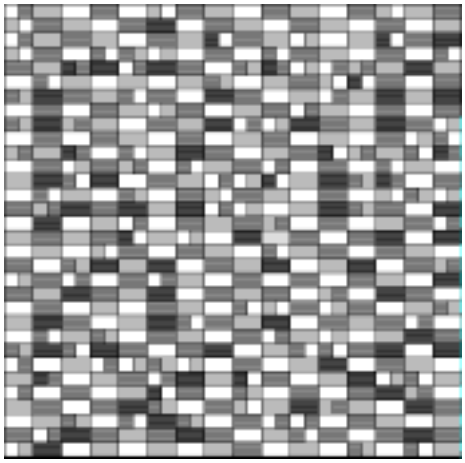

Figure 6: Probabilistic uniform partitioning

The complete split levels produce only $2^{\lfloor \log_2 P \rfloor}$ pages, thus $P - 2^{\lfloor \log_2 P \rfloor}$ additional regions are needed to obtain the given number of pages, i.e., the table size. These regions are created from the $2^{\lfloor \log_2 P \rfloor}$ pages by splitting these pages with respect to attribute j. Therefore the probability of an additional split in an attribute j is:

$$\text{probability}_j(d,P) = \begin{cases} \frac{P}{2^{\lfloor \log_2 P \rfloor}} - 1 & \text{, if } j = \text{probabilistic}(d,P) \\ 0 & \text{, otherwise} \end{cases}$$

If the probability of an incomplete split is taken into account, the number of slices overlapped by a query range in a certain dimension can be derived from the value for the completed splits. By subtraction we calculate, how many slices would be overlapped additionally, if another completed split were existing. For each of these splits the probability of its existence is $\text{probability}_j(d, P)$. The average number of additional splits may then be calculated by multiplication.

Thus, the average number of slices in dimension j overlapped by the range $[y_j, z_j]$ is:

$$n_j(d, P, y_j, z_j) = n(y_j, z_j, l_j(d, P)) + (n(y_j, z_j, l_j(d, P) + 1) - n(y_j, z_j, l_j(d, P))) \cdot \text{probability}_j(d, P)$$

The key attributes are independent and the query box $[[x, y]]$ is iso-oriented with respect to each dimension. Thus the total number of pages is obtained by multiplication of the slices in each dimension as in the previous sections.

Figure 7: Range Queries and Probabilistic uniform partitioning

4.4 Cost-Functions and Selectivity

For independently uniformly distributed data the restriction of each attribute in percent of space also defines the selectivity of that attribute. Thus the restriction $[y_j, z_j]$ in attribute A_j has a selectivity of $s_j = \hat{z}_j - \hat{y}_j$.

$\prod n_j(d, P, y_j, z_j)$ can be considered to be $\prod \hat{s}_j \cdot P$ with $\hat{s}_j$ as a special ceiling function rounding the selectivity s_j of dimension j to the next partitioning grid point. Then the cost function can also be represented as:

$$c(d,P,s) = \prod_{j=1}^{d} n_j(d,P,0,s_j) = P \cdot \prod_{j=1}^{d} \hat{s}_j$$

Note that by using 0 and s_j instead of z_j and y_j some information for the accuracy of the cost function is lost, since the position of the query box influences the number of Z-regions overlapped by a query box. Thus $c(d, P, y, z)$ should be used instead of $c(d, P, s)$, if not only the selectivity, but also start and endpoint of the query in space are known.

5. Conclusion and Future Work

We have derived a cost model for range queries for UB-Trees storing uniformly distributed data with independent dimensions. Next to analyzing and predicting the results of range queries, our results may also be used in conjunction with the Tetris algorithm and thus also take sort operations into account. Due to a lack of space the conclusion, a more detailed elaboration of the cost model as well as the conclusions for the Tetris paper can be found in [Mar99]. There the interested reader may also find performance comparisons using the cost model as well as comparisons of the cost model with the actual query performance.

We found out that our cost model is a sufficient decision basis for query optimization under the assumption of uniformly distributed, independent attributes. However, our cost model may also be generalized to arbitrary data distributions by taking the selectivity of each dimension into account. In this case the major importance comes to the application of techniques like multidimensional histograms for storing and maintaining the multidimensional data distribution. A very accurate estimate here might be to use the actual UB-Tree nodes and levels as a histogram in the same way as B-Trees can be considered to be histograms.

The cost model can also be applied to access methods that specifically deal with the data distribution, like VUB-Trees [MBB99]. The cost model is applicable for all databases that store multidimensional point data, e.g, data warehousing applications, in particular for the TPC-D benchmark (cf. [MZB99] for an evaluation), archiving systems, life-cycle management, data mining and the like.

Since any relation can be considered to store multidimensional points, it even generalizes to multi-attribute access methods for relational database systems. Our cost models can be used for cost-based query optimization as well as for simulation, theoretical comparison and analysis of access methods. In addition, it is useful for run-time prediction of algorithm, e.g., when informing a user about the expected response time to his query or when balancing system load.

Our future work includes deriving a set of cost based decision rules for how to – possibly multidimensionally - index a relation for a given set of queries. We also intend to apply our model to cost based query optimization and combine the model with multidimensional histograms in order to take dependencies and correlations between the attributes into account.

Acknowledgments

We thank our project partners Teijin Systems Technology, the European Commission and Microsoft Research for funding this research work.

References

[AS94] W. G. Aref and H. Samet: *A Cost Model for Query Optimization Using R-Trees.* ACM-GIS 1994

[AV88] A. Aggarwal and J.S. Vitter. *The Input/Output Complexity of Sorting and Related Problems.* Comm. ACM 31(9), p.1116 – 1127, 1988

[Bec93] L. Becker: *A New Algorithm and a Cost Model for Join Processing with Grid Files.* GI Datenbank Rundbrief 12, 1993

[BBK+97] S. Berchtold, C. Böhm, D. A. Keim et al. *A Cost Model For Nearest Neighbor Search in High-Dimensional Data Space*. PODS 1997, p. 78-86

[Bay97] R. Bayer. *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, March, 1997.

[CNP99] P. Ciaccia, A. Nanni, and M. Patella: *A Query-sensitive Cost Model for Similarity Queries with M-tree*. Australasian Database Conference 1999, p. 65-76

[Dat88] C.J. Date. *A Guide to the SQL Standard*,2nd edition. Addison-Wesley, 1988.

[GGS96] S. Ganguly, A. Goel, and A. Silberschatz: *Efficient and Acurate Cost Models for Parallel Query Optimization*. PODS 1996, p. 172-181

[GG97] V. Gaede and O. Günther. *Multidimensional Access Methods.* ACM Computing Surveys 30(2), 1997.

[GR97] J. Gray and A. Reuter. *Transaction Processing: Concepts and Techniques. 3rd edition*. Morgan Kaufmann Publishers, 1997.

[Gra93] G. Graefe. *Query Evaluation Techniques for Large Databases*. ACM Computing Surveys 25, 1993, pp. 73-170.

[Gre89] D. Greene. *An Implementation and Performance Analysis of Spatial Data Access Methods*. Proc. of 5th ICDE, 1989.

[HAM+97] C.T. Ho, R. Agrawal, N. Megiddo, and R. Srikant. *Range Queries in OLAP Data Cubes*. Proc. of ACM SIGMOD Conf., Tucson, Arizona, 1997, pp. 73-88.

[HR96] E.P. Harris and K. Ramamohanarao. *Join algorithm costs revisited*. VLDB Journal, 5, 1996.

[Knu73] D.E. Knuth. *The Art of Computer Programming Volume 3: Sorting and Searching*. Addison-Wesley, Reading, MA, 1973.

[Mar99] V. Markl. *MISTRAL: Processing Relational Queries using a Multidimensional Access Method*. Ph.D. Thesis, Technische Universität München, 1999

[MBB99] V. Markl, M. Bauer and R. Bayer. *Variable UB-Trees*. DMDW 99, Magdeburg, 1999

[Mer81] T.H. Merret. *Why Sort-Merge gives the best Implementation of the Natural Join*. ACM SIGMOD Record 13(2), 1981, pp. 39-51.

[MZB99] V. Markl, M. Zirkel, and R. Bayer. *Processing Operations with Restrictions in Relational Database Management Systems without external Sorting*. Proc. of ICDE, Sydney, Australia, 1999.

[RMF+00] F. Ramsak, V. Markl, R. Fenk et al. *Integrating the UB-Tree into a data-base system kerne*l. VLDB 2000, Cairo.

[ROH99} M. Tork Roth, F. Ozcan, and L: M. Haas: *Cost Models DO Matter: Providing Cost Information for Diverse Data Sources in a Federated System*. VLDB 1999, p. 599-610

[TSS00] Y. Theodoridis, E. Stefanakis, and T. K. Sellis: Efficient Cost Models for Spatial Queries Using R-Trees. TKDE 12(1), p. 19-32, 2000

[Ull88] J.D. Ullman. *Database and Knowledge Based Systems Volume I*. Computer Science Press, Rockville, MD, 1988.

[Ull89] J.D. Ullman. *Database and Knowledge Based Systems Volume II*. Computer Science Press, Rockville, MD, 1989

[Vit99] J.S. Vitter. External Memory Algorithms and Data Structures. DMACS Series in Discrete Mathematics and Theoretical Computer Science, 1999