Optimizing Physical design of Multidimensional Files for Join Queries

Salahadin Mohammed
Dept. of Info. & Comp. Sci.
KFUPM
KFUPM Box 1853
Dhahran 31261, Saudi Arabia
adam@ccse.kfupm.edu.sa

Bala Srinivasan
School of Comp. Sci. & SW Eng.
Monash University
Victoria 3168
Australia
bala.srinivasan@infotech.monash.edu.au

ABSTRACT
Optimally organizing multidimensional data is NP-hard. The little work that has been done in optimising multi-dimensional data was limited to uniform data distribution and rarely considered the probability of use of each query. And those who did consider the probability of use of each query, they were limited to either partial match query or range query. This work shows that by combining heuristics and combinatorial algorithms, near-optimal solutions can be found which organize multidimensional data (uniform or skewed) on which join queries are efficiently performed. The experimental results of the proposed algorithms show that performance gains of up to 716% are achieved, when compared with standard schemes. Moreover, the proposed algorithms are not very sensitive to the change in the query distribution. The result show that if the query probabilities change by up to 80% of their original values, the original storage organizations remain near optimal.

KEY WORDS
Join, Multidimensional files, choice vector.

1 Introduction

The time taken to answer a query is mainly measured by the number of disk accesses performed to retrieve the records described by the query. If the described records of a query are scattered in many blocks, the cost of the query will be high. But if the described records are clustered in smaller number of blocks the cost of the query will be lower.

The join operation is frequently used and is one of the most time consuming and data-intensive operations in relational query processing. It is used to combine tuples from two or more relations based on a specified condition known as *join-condition*. Tuples of the input relations are combined when they satisfy the specified join-condition. The result of a join operation is a relation which has some or all of the attributes of the input relations.

To lower the average query cost, related records must be clustered in fewer blocks. To cluster records described by a query, each record in a file is placed using the values of one or more of its attributes. The attributes that determine the placement of records in a file are called the *organizing attributes*. A file whose records are placed using one organizing attribute has a uni-dimensional file organization. B-trees [11]is an example of uni-dimensional file organization.

A file whose records are placed using more than one organizing attribute has a *multidimensional file organization*. Some examples of multidimensional file organization are X-tree, BV-tree, hP-tree and many other [2].

With an increasing number of applications such as computer aided design, VLSI, robotics, geometric or geographic systems, medical imaging, environmental protection, data warehouse, visual perception and text retrieval systems, searching using several attributes is more common than using one attribute.

Few researchers studied techniques of minimising query costs when using multidimensional file structures, but nearly all of them were limited to uniform data distributions [5–8]. To our knowledge this is the first study which introduces techniques of optimally clustering records in a multidimensional file structure when the data distribution is skewed.

This paper has 5 sections. Section 2 gives introduction to multidimensional files. The proposed join algorithm together with the heuristic and combinatorial algorithms used in the optimisation of physical database design are discussed in Section 3. Results and analysis of the proposed algorithms are discussed in Section 4. Section 5 is the conclusion.

2 Nested and Balanced Grid (BANG) File

The Multidimensional file (MDF) that we used in testing the proposed solutions is the BANG file [3]. A relation, R_i,

which has n organizing attributes, $A_{i,0}, A_{i,1}, \ldots, A_{i,n-1}$, (organized as a BANG file) can be envisioned as an n-dimensional data space where each dimension corresponds to the domain of an attribute in the relation. The *domain-space* of the relation is the cartesian product of $D_{i,0}, D_{i,1}, \ldots, D_{i,n-1}$, where $D_{i,j}$ is the domain of $A_{i,j}$. A record is represented as a point and a block as an n-dimensional partition within the domain-space.

BANG uses hashing scheme to place its records. In hashing schemes, the address of a disk block where a record resides is determined by a hash key calculated for that record. If the file on which the record resides has one organizing attribute, a hash function is applied to the value of that attribute. But if the file has many organizing attributes, then as many hash functions are used. Each organizing attribute has a hash function which maps a value into bit strings. For example, in a relation, R_i, with organizing attributes $A_{i,0}, A_{i,1}, \ldots A_{i,n-1}$, n hash functions, $h_{i,0}, h_{i,1}, \ldots h_{i,n-1}$, are employed. $h_{i,j}$, maps each $A_{i,j}$ value to a bit string, $b_{i,j,0} b_{i,j,1} \ldots b_{i,j,c_{i,j}-1}$, where $c_{i,j}$ is the minimum number of bits needed to represent any values of $A_{i,j}$. The hash key for a record is constructed by taking $d_{i,j}$ bits, where $0 \leq d_{i,j} < c_{i,j}$, from the bit string of $h_{i,j}$ and combining them in a specific order. This order is maintained by a structure known as a *choice vector*. In short a choice vector specifies the order by which the hashed bit strings are combined to form a hash key of a record. Each element of a choice vector is a bit position and is denoted as $b_{i,j,k}$, where $0 \leq k < c_{i,j}$.

3 The Proposed Join Algorithm

The proposed join algorithm have two main modules, a *selection-module* and a *matching-module*. define

3.1 The Selection-module

The selection-module exploits the partitioning and clustering properties of the BANG file in selecting the next set of join-compatible partitions (partitions whose join attribute values overlap). For example, assume the following query, q_1, which uses R_0 and R_1 of Figure 1 as input relations.

SELECT $A_{0,0}$**,** $A_{0,1}$**,** $A_{1,1}$
FROM R_0**,** R_1 (q_1)
WHERE $A_{0,0}$ = $A_{1,0}$

In processing q_1, there is no need to matching $P_{0,3}$ and $P_{1,12}$ for join. They don't share tuples which can satisfy the join-condition because their join-attribute values don't overlap. The join-attribute values of $P_{0,3}$ are between 0 and 25 and that of $P_{1,12}$ are between 75 and 100. But

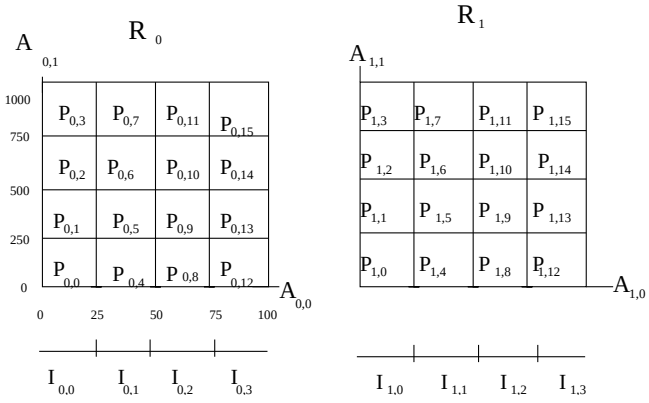

Figure 1. $P_{0,0}$ **and** $P_{1,0}$ **are join-compatible while** $P_{0,0}$ **and** $P_{1,12}$ **are not.**

$P_{0,3}$ and $P_{1,0}$ are join-compatible partitions, because their join-attribute values overlap. Both are between 0 and 25. $P_{0,3}$ is also join-compatible to $P_{1,1}$, $P_{1,2}$ and $P_{1,3}$. In fact each of $P_{0,0}$, $P_{0,1}$, $P_{0,2}$ and $P_{0,3}$ are join-compatible to $P_{1,0}$, $P_{1,1}$, $P_{1,2}$ and $P_{1,3}$. So the selection-module identifies such join-compatible partitions and passes them to the matching-module. The matching-module matches the tuples of these partitions for join. Once the join of the current join-compatible set is completed, the algorithm again starts selecting the next join-compatible set. In case the of q_1, the next join-compatible set contain partitions $P_{0,4}$, $P_{0,5}$, $P_{0,6}$ and $P_{0,7}$ with $P_{1,4}$, $P_{1,5}$, $P_{1,6}$ and $P_{1,7}$.

Before the start of the two modules, the join algorithms compute the number of join-compatible sets. This is done by logically dividing the join-attribute domain into a number of equal *intervals*. Then partitions are mapped into these intervals. Partitions mapping to the same interval form a set called a *wave*. The selection-module uses intervals and waves to select the next join-compatible partitions.

3.1.1 Intervals

Before performing selection or matching, the join-attribute domain is logically dividing into a number of equal *intervals*. The number of intervals can be computed from the choice vector. In a choice vector, the number of elements which correspond to a join-attribute $A_{i,j}$ is $d_{i,j}$, then the number of intervals along $D_{i,j}$ is $2^{d_{i,j}}$. The size of each interval is equal to the size of join-attribute edge of the smallest partition in the relation. In case of q_1, the domain of the join-attribute is partitioned into 4 equal intervals as shown in Figure 1. This is because the size of the smallest join-attribute edge in the relation is a fourth of the join-attribute domain size, which is 100.

For now us assume that the number of intervals in the input relations are equal, and each partition be within one

interval. Later in this section we will remove these restrictions.

Intervals are labeled. Intervals corresponding to R_i are labeled as $I_{i,0}, I_{i,1}, I_{i,2}$ and so on. The number of intervals can be computed from the choice vector. In a choice vector, the number of elements which correspond to a join-attribute $A_{i,j}$ is $d_{i,j}$, then the number of intervals along $D_{i,j}$ is $2^{d_{i,j}}$. Attribute values in $I_{i,j+1}$ are higher than those in $I_{i,j}$. For example, in Figure 1 the intervals of R_0 are labeled as $I_{0,0}, I_{0,1}, I_{0,2}$, and $I_{0,3}$. $I_{0,0}$ covers attribute values between 0 and 24 inclusive, $I_{0,1}$ covers values between 25 and 49 inclusive, $I_{0,2}$ covers values between 50 and 74 inclusive, and $I_{0,3}$ covers values between 75 and 99 inclusive.

3.1.2 Waves

Partitions of a relation whose join-attribute edge overlapping the same interval belongs to the same join-compatible set. For example, partitions overlapping $I_{i,k}$ are put in one set and partitions overlapping $I_{i,k+1}$ are put in a separate set. We call such a set a *wave*.

Waves are labeled as $W_{i,0}, W_{i,1}, W_{i,2}$ and so on. There is one-to-one mapping between waves and intervals. $W_{i,k}$ contains only partitions overlapping $I_{i,k}$. This makes the number of waves to be equal to the number of intervals.

The join-attribute edge of a partition can span more than one interval. Such a partition can become a member of more than one wave. Some times all the members of a one wave are also members of another wave. This happens when some partitions of a wave span more than one interval.

When the number of join-attributes in a relation is more than one, the number of intervals (waves) created is a multiple of all the individual join-attribute intervals.

3.2 The Matching-module

The matching-module is performed in a nested loop. In the outer loop, the partitions of one of the two waves are read and their tuples are put into a hash table. In the inner loop, the partitions of the other wave are read and their tuples probed into the hash table for join with the tuples of the outer loop.

3.3 Wave Size and Choice Vector

Choice vectors significantly affect the cost of a join query. For example, assume the join of two relations R_0 and R_1 of Figure 1 with $A_{0,0} = A_{1,0}$ as a join-condition. Let us assume that both $A_{0,0}$ and $A_{1,0}$ have the same domain ranging between 0 and 100, and the first-wave always belongs to $A_{0,0}$ and the second-wave to $A_{1,0}$. Let the buffer size allocated for the first-wave be 2 and for the second-wave be 1 and for the result be 1. Then the cost of each join-compatible wave is as follows: $4 + \lceil \frac{4}{2} \rceil \times 4 = 12$. Since we have 4 join-compatible waves, the join of R_0 and R_1 will cost $4 \times 12 = 48$ disk accesses. But if $A_{1,0}$ of R_1 was partitioned into 8 intervals instead into 4 (using a different choice vector) and $A_{1,1}$ was partitioned into 2 intervals instead into 4, then the cost would have been 32. This is because the corresponding waves are smaller.

So allocating more elements of a choice vector to join-attributes results in smaller size waves. The problem now is, given a number of join queries and their probabilities, to find optimal choice vectors which results in minimal average cost. Finding optimal choice vectors for arbitrary query distribution is NP-hard [9]. Hence we will use heuristic algorithms and combinatorial optimisation techniques to find optimal or near optimal choice vectors.

3.4 Combinatorial Optimisation Techniques

To attempt to find good choice vector, we used heuristic and combinatorial optimisation algorithms. These techniques are not guaranteed to find optimal choice vector to any problem. In the subsequent sections we use the term *minimal* or *optimised* to indicate a solution which is the result produced by one or more of these techniques. These minimal solutions are typically local minima or local optima.

The optimisation solutions that we used can be described in the following way. Consider a set, $\hat{k}$, of n non-negative integers, k_i, upon which cost function, f, is defined as [4]

$$C = f(\hat{k})$$

where $\hat{k} = \{k_0, k_1, \ldots, k_{n-1}\}$. The main objective here is to find $\hat{k}_{min}$, such that $f(\hat{k}_{min}) < f(\hat{k})$, for all members of $\hat{k}$. The constraint

$$\sum_{i=0}^{n-1} k_i = k$$

must be satisfied.

Relating this to a choice vector, k is the number of elements in the choice vector, k_i is the number of bits allocated to the ith attribute, $\hat{k}$ is a bit allocation, and $\hat{k}_{min}$ is the optimal bit allocation.

3.4.1 Simulated Annealing

There are many heuristic and combinatorial optimisation algorithms such algorithms are minimal marginal increase and simulated annealing [1]. Nurmela in [10] found that simulated annealing typically performed as well or better than a number of other combinatorial optimisation methods. He also found that simple genetic algorithms which did not use problem specific knowledge did not perform as well as

local search algorithms. Simulated annealing is a class of optimisation algorithms based on Monte Carlo techniques.

The algorithm begins by selecting a random choice vector and computes the cost of the problem on hand using cost functions, which are dependent on the choice vector. Then in each iteration, the algorithm computes new choice vectors and accepts the new choice vector as the basis for further perturbations if it improves the cost or when a *cooling function* determines that it be accepted. The cooling function is a monotonically decreasing function which specifies the probability of accepting a choice vector which does not improve the cost. In the early iteration the probability of accepting a choice vector that does not improve the cost function is high, but approaches to zero in the later stages. There are a number of parameters which can be used to control the amount of computational resources used by the algorithm. The algorithm terminates when the costs has not improved after pre-specified number of iterations since the last accepted choice vector.

3.4.2 Cost Functions

This section discusses the cost functions that we used together with simulated annealing to find optimal choice vectors.

The average join query cost is:

$$C_Q = \sum_q C_q \times p_q \tag{1}$$

where p_q is the probability of q and Q is a set of join queries. C_q is the cost of a single query. The cost of joining two join compatible waves is:

$$\begin{aligned} C_{W_{\bar{k}}} &= \min(|W_{i,k}|, |W_{j,\bar{k}}|) + \\ &\quad \max\left(|W_{i,k}|, |W_{j,\bar{k}}|\right) \left\lceil \frac{\min(|W_{i,k}|, |W_{j,\bar{k}}|)}{B} \right\rceil \end{aligned} \tag{2}$$

Where *B* is the buffer size.

Let the choice vector elements of R_i be d_i out of which $d_{i\star}$ belong to the join attribute. Similarly let the choice vector elements of R_j be d_j and that of the its join attribute be $d_{j\star}$. The number of waves created in R_i and R_j is 2^{d_i} and 2^{d_j} respectively. So on the average each wave of R_i will contain $2^{d_i - d_{i\star}}$ and each wave of R_j will contain $2^{d_j - d_{j\star}}$. Let $\delta_i = d_i - d_{i\star}$ and $\delta_j = d_j - d_{j\star}$. Hence Equation 2 can be rewritten as:

$$\begin{aligned} C_{W_{\bar{k}}} &= \min(2^{\delta_i}, 2^{\delta_j}) + \\ &\quad \max(2^{\delta_i}, 2^{\delta_j}) \left\lceil \frac{\min(2^{\delta_i}, 2^{\delta_j}))}{B} \right\rceil \end{aligned} \tag{3}$$

The cost of a query is equal to the cost of its join compatible joins and is represented as:

$$C_q = \sum_{k=0}^{2^{\min(\delta_i, \delta_j)} - 1} C_{W_{\bar{k}}} \tag{4}$$

By combining Equations 1, 4 and 3 we end up with:

$$\begin{aligned} C_Q &= \sum_q p_q \sum_{k=0}^{2^{\min(\delta_i, \delta_j)} - 1} \min(2^{\delta_i}, 2^{\delta_j}) + \\ &\quad \max(2^{\delta_i}, 2^{\delta_j}) \left\lceil \frac{\min(2^{\delta_i}, 2^{\delta_j})}{B} \right\rceil \end{aligned} \tag{5}$$

So by using simulated annealing together with Equation 5 optimised choice vectors can be found for R_i and R_j.

4 Results and Analysis

In this section we present three sets of experimental results comparing the performance of the optimised and cyclic choice vectors. In a cyclic choice vector equal number of bits are chosen from each organizing attribute and they are arranged in a circular fashion.

The first set of results shows the performance of the optimised and the cyclic choice vectors on different data and query distributions. The second set of results shows the effect of the number of attributes on the performance of both choice vectors.

Query distributions change over time. A choice vector optimised for one query distribution may not perform as well if the query distribution changes. A solution, based on simulated annealing and Equation 5, is called *stable* if a slight change in the query distribution doesn't affect the optimality of the solution. The last set of results demonstrate the stability of the optimised choice vector.

4.1 Environment

We implemented a BANG file with our extension of using a choice vector during partition splitting. In each experiment we used randomly generated queries and assigned each of them a randomly generated probability. Unless specified, we used a page size of 1024 bytes, four integer attributes per record and one million randomly generated records per relation (BANG file). We ran all our experiments on a SPARC station 20.

The data distributions used were uniform, clustered regions, a linear correlation, and a non-linear correlation function (a sine wave). Examples of these are shown in Figure 2.

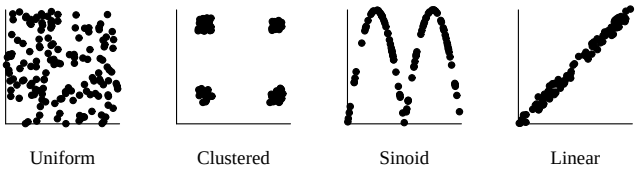

Figure 2. The four data distributions used in generating the results.

We refer to them as uniform, clustered, linear and sinoid. The number of intervals can be computed from the choice vector. In a choice vector, the number of elements which correspond to a join-attribute $A_{i,j}$ is $d_{i,j}$, then the number of intervals along $D_{i,j}$ is $2^{d_{i,j}}$., respectively.

In the experiments, four sets of query distributions were used. Query distributions in each set were generated randomly using a fixed set of seeds. The seeds used for one set were different from that of the other. In this paper, these four sets of query distributions are referred to as Θ_1, Θ_2, Θ_3 and Θ_4.

4.2 Effect of Data Distribution

The effect of using the optimised and cyclic choice vectors on the average query cost using different data and query distributions is shown in Tables 1 to 4. The first column in each of these tables shows the query distribution used. The second and the fourth columns correspond to the cyclic choice vector and show the cost in number of disk page accesses (DPA) and in lapse time, respectively. Similarly, the third and fifth columns show the costs corresponding to the optimised choice vector. The improvement in the number of DPA and time taken when using the optimised choice vector rather than the cyclic choice vector is shown in the final two columns.

In all the experiments performed the optimised choice vector performed better than the cyclic choice vector. The improvement is lower when both input relations have skewed data distributions. The more skewed the data distribution is, the more are the elements of the choice vector. More elements in the choice vector causes more number of waves hence less number of partition per wave which can fit into the available memory.

4.3 Effect of Number of Attributes

As the number of attributes increases, the number of attributes that are specified in few or no queries (*nonsignificant* attributes) is likely to increase. As the ratio of the nonsignificant to significant attributes increases, the performance improvement of the optimised choice vector over the cyclic choice vector decreases. This is because when the

Query Dist.	DBA		Time (sec)		Improvement in	
	Cyl	Opt	Cyc	Opt	DPA	Time
Θ_1	154634	60093	1438	589	2.57	2.44
Θ_2	151735	61208	1382	588	2.48	2.35
Θ_3	160048	60128	1483	594	2.66	2.50
Θ_4	162986	58960	1464	554	2.76	2.64

Table 1. Average query cost for a uniform data distribution.

Query Dist.	DBA		Time (sec)		Improvement in	
	Cyl	Opt	Cyc	Opt	DPA	Time
Θ_1	153329	62549	1397	572	2.45	2.41
Θ_2	150170	60740	1381	573	2.47	2.41
Θ_3	159020	60575	1351	529	2.62	2.55
Θ_4	161434	58211	1402	517	2.77	2.71

Table 2. Average query cost for a clustered data distribution.

Query Dist.	DBA		Time (sec)		Improvement in	
	Cyl	Opt	Cyc	Opt	DPA	Time
Θ_1	51684	50073	477	468	1.03	1.02
Θ_2	51462	49133	480	466	1.05	1.03
Θ_3	61156	59429	558	552	1.03	1.01
Θ_4	51756	48356	472	441	1.07	1.04

Table 3. Average query cost for a sinusoidal data distribution.

Query Dist.	DBA		Time (sec)		Improvement in	
	Cyl	Opt	Cyc	Opt	DPA	Time
Θ_1	52213	49131	482	459	1.06	1.05
Θ_2	52040	48759	485	458	1.07	1.06
Θ_3	55847	51415	490	449	1.09	1.09
Θ_4	52477	48177	474	439	1.09	1.08

Table 4. Average query cost for a linear data distribution.

cyclic choice vector is used, peer-splitting based on the non significant attributes increases. As a result, the performance improvement achieved by using the optimised choice vector instead of the cyclic choice vector increases. Our experimental results showed that, on the average, the gains were 1.05, 1.62, 2.62, and 7.16 when 2, 3, 4, and 8 attributes were used respectively.

5 Conclusion

Our study shows that given a probability distribution of join queries, an efficient physical database design can be created by using simulated annealing and Equation 5. Unlike previous approaches, our approach is not limited to a uniform data distribution or to independently specified attributes, and the precise nature of any non-uniformity does not need to be known.

When compared to the cyclic choice vector, our results show that the optimised choice vector produces more efficient physical database designs, reducing the average query cost by up to 716%.

The improvement gained by using an optimised choice vector instead of the cyclic choice vector increases as the number of attributes increases. This is because the elements of the insignificant attributes in the cyclic choice vector is higher.

There is no need to rearrange the optimised choice vector whenever the query distribution changes by a small amount. Our experiments show that the optimised choice vector built for the current query distribution will remain almost as good as the optimised choice vector built for a variation on the current query distribution even when the query distribution changes by 80%. There was a degradation of less than 5% in performance.

We conclude that if the query distribution is known and a file structure which evenly distributes records amongst disk pages is used regardless of the data distribution, an optimised choice vector produces an efficient physical database design.

Acknowledgements: This work has been supported by King Fahd University of Petroleum and Minerals.

References

[1] E. Aarts and J. Korst. *Simulated annealing and Boltzmann Machines*. Wiley, 1989.

[2] Jiyuan An, Hanxiong Chen, Kazutaka Furuse, Masahiro Ishikawa, and Nobuo Ohbo. The complex polyhedra technique: An index structure for high-dimensional space. In Xiaofang Zhou, editor, *Thirteenth Australasian Database Conference (ADC2002)*, volume 5 of *Conferences in Research and Practice in Information Technology*, pages 33–40, Melbourne, Australia, 2002. ACS.

[3] M. Freeston. The bang file: A new kind of grid file. In U. Dayal and I. Traiger, editors, *The ACM SIGMOD International Conference on Management of Data*, pages 260–269, San Francisco, California, USA, May 1987.

[4] E. P. Harris. *Towards optimal storage design for efficient query processing in relational database systems*. PhD thesis, The University of Melbourne, Melboure, Australia, 1994.

[5] E. P. Harris and K. Ramamohanarao. Generalising minimal marginal increase to cluster records in multi-dimensional data files. In John Roddick, editor, *Database Systems '1999, Proceedings of the 10th Australasian Database Conference*, pages 129–140, Auckland, New Zealand, January 1999. Springer.

[6] Y. E. Ioannidis and Y. C. Kang. Randomized algorithms for optimizing large join queries. In *Proceedings of the 1990 ACM SIGMOD International Conference on the Management of Data*, pages 312–321, Atlantic city, New Jersey, USA, May 1990.

[7] R. S. G. Lanzelotte, P. Valduriez, and M. Zait. On effectiveness of optimisation search strategies for parallel execution spaces. In *Proceedings of the 19th VLDB Conference*, pages 493–504, Dublin, Ireland, August 1993.

[8] J.-H. Lee, Y.-K. Lee, K.-Y. Whang, and I.-Y. Song. A region splitting strategy for physical database design of multidimensional file organizations. In *Proceedings of the 23rd VLDB Conference*, pages 416–425, Athens, Greece, August 1997.

[9] S. Moran. On the complexity of designing optimal partial-match retrieval systems. *ACM Transactions on Database Systems*, 8(4):543–551, December 1983.

[10] K. J. Nurmela. Constructing combinatorial designs by local search. Department of Compuer Science, Digital Systems Laboratory A-27, Helsinki University, Finland, November 1993.

[11] Beng Chin Ooi and Kian Lee Tan. B-trees: Bearing fruits of all kinds. In Xiaofang Zhou, editor, *Thirteenth Australasian Database Conference (ADC2002)*, volume 5 of *Conferences in Research and Practice in Information Technology*, pages 13–20, Melbourne, Australia, 2002. ACS.