**Title**
PHYSICAL DESIGN OF TEMPORAL DATABASES



**Authors**
Rotem, D.
Segev, A.

**Publication Date**
1986-06-01

# Lawrence Berkeley Laboratory
UNIVERSITY OF CALIFORNIA, BERKELEY

## Information and Computing Sciences Division



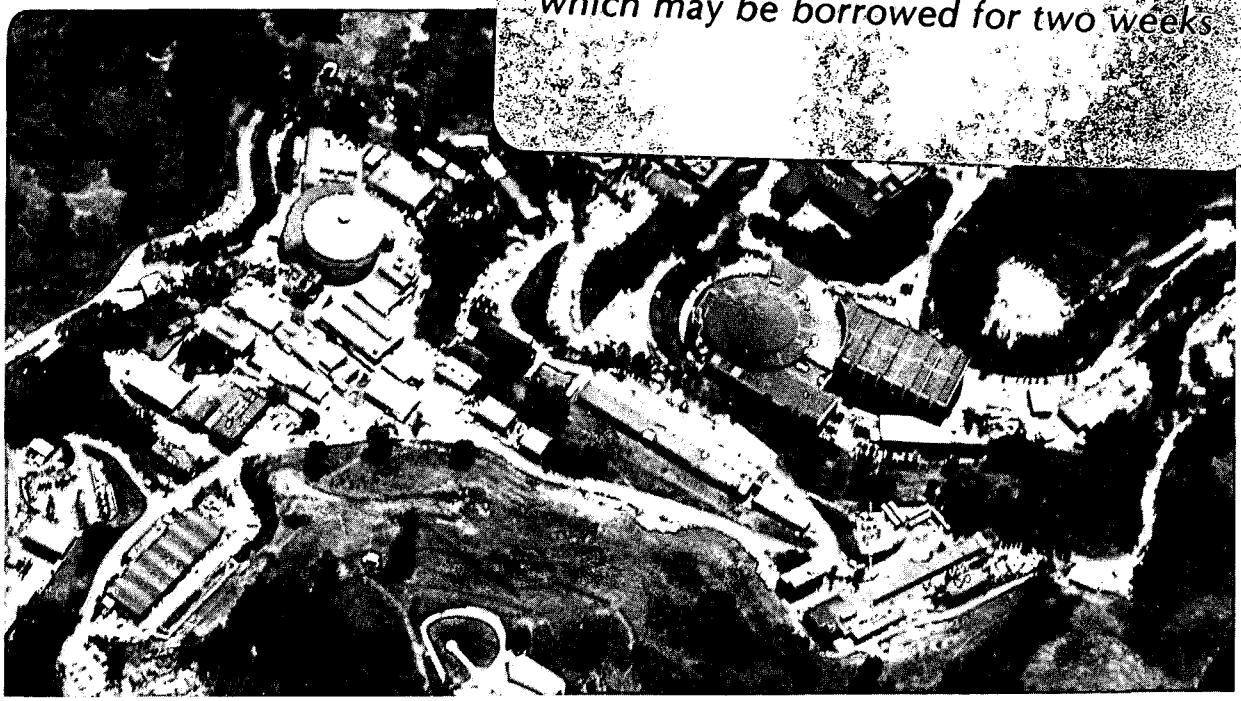
To be presented at the Third International Conference on Data Engineering, Los Angeles, CA, February 2-6, 1987

PHYSICAL DESIGN OF TEMPORAL DATABASES

D. Rotem and A. Segev

June 1986





Prepared for the U.S. Department of Energy under Contract DE-AC03-76SF00098

## DISCLAIMER

LBL-21747

# Physical Design of Temporal Databases

Doron Rotem† and Arie Segev‡

†Computer Science Research Department
University of California
Lawrence Berkeley Laboratory
Berkeley, California 94720

‡School of Business Administration
The University of California
Berkeley, CA 94720

June, 1986

This research was supported by the Applied Mathematics Sciences Research Program of the Office of Energy Research, U.S. Department of Energy under contract DE-AC03-76SF00098.

# PHYSICAL DESIGN OF TEMPORAL DATABASES

*Doron Rotem*[†] *and Arie Segev*[‡]

† Lawrence Berkeley Laboratory
Department of Computer Science Research
The University of California
Berkeley, CA 94720

‡ School of Business Administration
The University of California
Berkeley, CA 94720.

*ABSTRACT*

In this paper, we propose a partitioning scheme of files in a temporal database system, and develop algorithms to minimize the overflow associated with that scheme. The proposed partitioning is suitable for range queries and aggregation operations along the time dimension. Simulation results show that the partitioning algorithms proposed in this paper result in better performance than in the case of a grid-type partitioning.

## 1. Introduction

Recently there has been a significant increase in the amount of research in the area of temporal databases [ARIA84, LUM84, SNOD84, CLIF85, RUBE85, SNOD85, AHN86, SHOSH86, and others]. For a survey of the role of time in information processing, see [BOLO82]. The importance of incorporating the time dimension into database systems has long been recognized. The dramatic decrease in the cost/performance ratio of hardware components has made it feasible to support the time dimension online provided that efficient storage methods and access algorithms are developed. In this paper we deal with the case of temporal databases, be they commercial or scientific. Such databases are often characterized by their static nature (especially in the case of historical temporal data), and the importance of the time domain (these databases are frequently used for time-series analysis).

Most of the studies on the support of the temporal dimension have dealt with the conceptual [AHN86, SNOD85, SNOD84, CLIF85] rather than the physical [LUM84, RUBI85, SHOSH86] level. Like in any other database, it is clear that there is not a single file structure for a temporal database that universally dominates all others. The suitability of a particular file structure is dependent both on the data and on the access pattern. For example, the physical organization proposed in [LUM84] is appropriate for the case of querying the complete history of an attribute. In the case of temporal databases, the most important property of the data is its behavior with respect to time. Such a behavior can be clearly identified through the framework of viewing temporal data proposed by [SHOSH86]. The essence of that framework is to view temporal data as a time sequence with certain properties. These properties have a major impact on the physical organization of the files.

It is our view that in many instances, a multidimensional partitioning of the file is the appropriate approach to organizing temporal data. In traditional databases, the problem of multidimensional file partitioning (MDFP) arises in many applications where it is required to store files which are indexed by one or more search attributes to disk pages such that the mapping from the key space to the physical address space is order preserving. Such an order-preserving map-

ping is important in an environment where the frequency of range queries and sequential access by key values is high. Examples of MDFP schemes are the Grid Files of different types [NIEV84, REGN85, OZKA85, OZKA85a], and the file structures described in [MERR78, MERR82, MERR84] . In all of these methods the possible range of values for each attribute is partitioned into segments, the intersection of these segments define hyper-rectangles or cells. Each tuple is associated with a cell based on the segments to which its attribute values belong. An MDFP structure is a three level hierarchical structure. The first level is a relatively small file with information about the partitioning points and is kept in fast storage. The second level is a directory with a single entry per each cell of the partitioning. This entry contains a pointer to the data page which contains the tuples associated with this cell. The last level is the actual file where the data tuples are stored. Typically the two last levels are resident on disk because of their size. A query against this structure is answered as follows: Using the attribute values specified by the query, the information kept in fast storage is used to compute the disk address of the relevant directory entry or entries on disk, these in turn contain pointers to the data pages where the tuples which form the answer to the query are stored.

In a given application environment, data files are classified as static or dynamic depending on the rate of insertion and deletion transactions. In recent years, most of the research has focused on dynamic files, resulting in several file organization methods, e.g., the grid file method [NIEV84]. Some work has also been done on static MDFP [MERR84]. For the temporal databases considered in this paper, a static algorithm is more appropriate especially when the database contains historical data for the purpose of analysis and decision making. In Section 6, we will indicate how static structures can be incorporated into a dynamic environment.

In Section 2, we explain the partitioning concept and discuss the parameters relevant to the decision on how to partition the file. In particular, we consider two types of partitioning: symmetric and asymmetric. Section 3 presents existing algorithms for the symmetric case, and in Section 4, we develop algorithms for the asymmetric case. The results of simulation experiments are reported in Section 5; these results demonstrate the advantage of the asymmetric partitioning in

terms of disk accesses. The paper is concluded with a summary and a discussion in Section 6.

## 2. Multidimensional Partitionings and Temporal Databases

As discussed in [SHOSH86], a temporal value is actually a triplet $(s, t, v)$, where $s$, $t$, and $v$ refer to the *surrogate*, *time* and *value* respectively. The time $t$ is the time to be managed and can be either logical or physical (see [LUM84] and [SNOD85] for a taxonomy of time). For an example, $(PROD1, JUNE, 1000)$ represents the fact that 1000 units of PROD1 were sold in June. We can represent the set of such triplets as a time sequence array ($TSA$) as illustrated in Figure 1 (adapted from [SHOSH86]). The set of values of surrogate $s_i$ can be viewed as a time sequence $TS_i$.

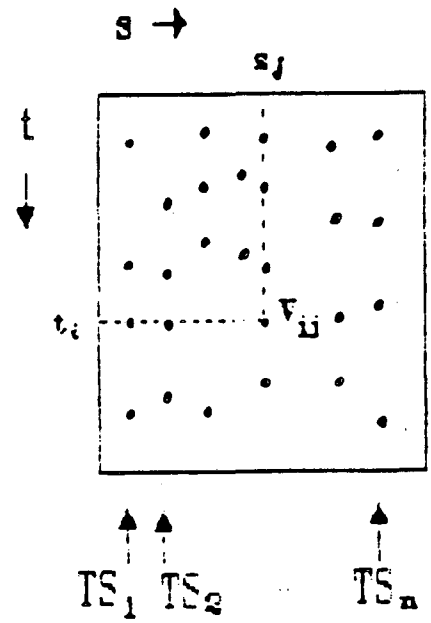



**Fig. 1: A Time Sequence Array**

We now explain the concept of multidimensional partitioning and show how it relates to temporal databases. As mentioned previously, in a MDFP scheme all the tuples which are associated with the same cell of the partitioning are stored on the same disk page. An overflow occurs when the number of tuples associated with a given cell exceeds the capacity of a disk page. We assume that there is a single common overflow area. In this paper we deal with the case of two search attributes (one of which is the time dimension), but the analysis can be easily generalized

to more than two dimensions. We will refer to an algorithm that determines the partitioning of the attribute space as a "partitioning algorithm". The parameters that determine the best partitioning are the size of the primary data area (in number of pages) - $K$ ; the page capacity (in number of tuples) - $c$ ; the storage utilization (or load factor) - $\alpha$; the number of unique values in the search attribute (to simplify the exposition, and without loss of generality, we assume that this number is the same for each search attribute) - $n$ ; and the frequency matrix $F=\{f_{ij}\}$, where $f_{ij}$ is the number of tuples having values i and j in the first and second search attributes, respectively. It should be noted that the matrix $F$ is different from the Time Sequence Array of Figure 1; in $F$, we store in position (i,j) the value $f_{ij}$ which represents the number of data tuples with values $t_i$ and $s_j$ in the time and surrogate attributes respectively. For example $f_{2,3}=5$ means that there are five transactions for surrogate $s_3$ at time unit $t_2$. The details of each such transaction is a record which will be kept on a data page. Note that all these five records (and possibly others associated with that cell) will reside on the same disk page. That is, the matrix $F$ serves as a basis for the allocation of data tuples to disk pages and the creation of an index to the data file. Our approach generalizes the one in [SHOSH86] in the sense that we also store the details of the transactions as records whereas the latter work only stores summary data about the number of sales or a single measurement per each entry. In our case the information given in $F$ is the input to partitioning algorithms which will determine which tuples will fall into the same cell and therefore will reside in the same disk page.

We consider two types of partitioning - symmetric and asymmetric. Figure 2 presents two examples of a symmetric partitioning. The matrix $F$ may represent daily sales transactions of a group of products. In this case, the first row of $F$ in Figure 2(a) has the following interpretation: the number of transaction records for products 1,2,3,4,5 in day 1 is 1,2,4,0,0 respectively. The second row represents the number of sales records in the second day, and so on. As can be seen from Figure 2, the matrix $F$ is partitioned vertically and horizontally by grid lines. The figure illustrates that the overflow can be reduced by changing the partitioning. Figure 3 presents an asymmetric partitioning of the same matrix. In this case, we partition $F$ vertically by grid lines,

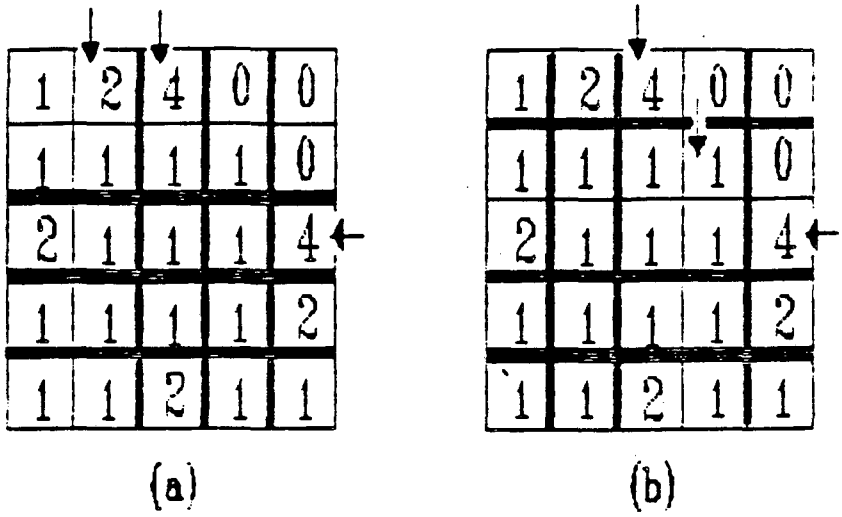



**Fig. 2: Two examples of a symmetric partitioning with $c=3$ and $K=16$. In (a) total overflow is 5 whereas in (b) it is 3. The arrows point to cells where overflow has occurred.**

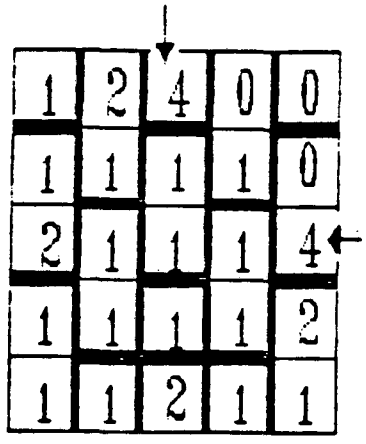



**Fig. 3: An example of an asymmetric partitioning which further reduces the overflow.**

and each vertical segment (the submatrix between two adjacent grid lines) is partitioned horizontally. As can be seen from Figure 3, the asymmetric partitioning helps to further reduce the amount of overflow.

The advantage of the symmetric partitioning over the asymmetric one is that it requires less storage space to keep the information about the partitioning (first level file), in this case we only need to keep the attribute values at which partitioning lines are inserted. In the asymmetric case we keep the same information as before about vertical lines but for each vertical segment we keep a list of its horizontal partitioning points. There is a difference between the two methods in the way in which data tuples are clustered in pages. In the symmetric method there is no priority of clustering by one attribute over another whereas in the asymmetric case we will give priority to cluster tuples together based on their surrogate value. We feel that with the reduction in the cost of main memory, the additional storage requirements of the asymmetric partitioning does not pose a serious problem. Moreover, the reduction in overflow outweighs the added cost of storage. The asymmetric partitioning is also appropriate in many instances of temporal data, where the dominant interest is in the attribute of an entity or a group of entities as a function of time; e.g., what were the sales of an item during a certain time period.

## 3. A symmetric partitioning

In this section, we present a symmetric-partitioning algorithm against which the asymmetric partitioning algorithm (see Section 4) will be compared. The symmetric-partitioning algorithm was developed in [ROTE86], where a detailed analysis of grid-type algorithms can be found. The following definitions will be used in the description of the algorithms presented in this paper.

Given a pair of integers $n_1$ and $n_2$, they are called permissible factors of $K$ if $n_1 \times n_2 = K$ *and* $n_i \leq n$ *for* $i = 1,2$. Let $P$ be the set of all permissible factor pairs of $K$. We can partition the matrix $F$ into $K$ cells by partitioning the rows into $n_1$ segments and the columns into $n_2$ segments. This is done by placing $n_1 - 1$ horizontal lines between the rows of $F$ and $n_2 - 1$ vertical lines between the columns of $F$. Let $C(F)$ and $R(F)$ be the two vectors obtained from $F$ in the following way: The vector $C(F) = < c_1, c_2, \ldots, c_n >$ is obtained from $F$ by setting $c_k = \sum_{i=1}^{n} f_{ik}$

for $1 \leq k \leq n$, i.e., each component $c_k$ is the sum of the elements of the $k^{th}$ column of $F$. Similarly, $R(F) = < r_1, r_2, \ldots, r_n >$ is the vector obtained by letting $r_i$ be the sum of elements in the $i^{th}$ row of $F$. Let $\{V, k, c\}$ denote a one dimensional partitioning problem of the vector $V$ into $k$ pages each with capacity $c$. The same notation will be used for the case of a two dimensional problem except for the following: $V$ will denote an $n$ by $n$ matrix and $k$ will be replaced by two factors. For each pair $n_1$, $n_2$ in $P$, we solve a pair of one dimensional problems $\{R(\dot{F}), n_1, n_2 \times c\}$ and $\{C(F), n_2, n_1 \times c\}$. Let $g_{n_1}$ and $g_{n_2}$ be the optimal solutions obtained for these two problems. By combining the horizontal and vertical line positions computed by the solutions $g_{n_1}$ and $g_{n_2}$, we obtain a solution $y(g_{n_1}, g_{n_2})$ to $\{F, n_1, n_2, c\}$.

The following theorem suggests a procedure for eliminating (fathoming) permissible factors from consideration.

**Theorem 1 ([ROTE86])**

Let $T$ be an optimal partitioning of the matrix $F$ into $K$ pages each with capacity $c$ with factors $n_1$ and $n_2$. Let $T'$ be an optimal (one dimensional) partitioning of $C(F)$ into $n_2$ pages each with capacity $n_1 c$. Let $TF(T)$ and $TF(T')$ denote the total overflow resulting from the partitioning $T$ and $T'$ respectively. Then, $TF(T') \leq TF(T)$.

□

We now describe an optimal solution procedure for the one-dimensional partitioning problem that will be used as a subroutine by the two-dimensional partitioning algorithms. The algorithm is based on the following relations:

Let $FL(i, j)$ denote the overflow incurred by allocating elements $\{f_i, \ldots, f_j\}$ to a segment.

$$FL(i, j) = Max \{0, (\sum_{t=i}^{j} f_t - c)\}$$

Let $TF(j, k)$ denote the total overflow incurred by optimally partitioning elements $\{f_1, \ldots, f_j\}$ into $k$ segments. The optimal solution to our problem is given by $TF(n, K)$. The recursive relation for the total overflow is given by:

$$TF(j, k) = \min_{i=k-1, \ldots, j-1} \{TF(i, k-1) + FL(i+1, j)\}, \quad j \geq k > 1$$

$TF(j,1) = FL(1,j)\ ,\ \ j \geq 1$

The above recursive equation is based on the well known 'principle of optimality' (see [HORO78]). In our case, this principle states that the optimal partitioning of the first $j$ elements of $F$ into $k$ segments must consist of an optimal partitioning of the first $i$ elements (for some $i$ such that $k-1 \leq i < j$ ) into $k-1$ followed by a single segment which contains all the remaining $j-i$ elements. The second equation is a straightforward boundary condition. Since efficient algorithms for solving the dynamic program given above are well-known, we will not explicitly state such an algorithm but simply refer to it as a solution to the one-dimensional problem.

Algorithm 1 below describes a heuristic procedure to partition the matrix $F$. Note that a partitioning of $F$ is equivalent to a partitioning of the attribute space and an assignment of data tuples to disk pages.

**Algorithm 1.**

1. For each element $(n_1,n_2)$ in $P$, solve the two one dimensional problems $\{C(F),n_2,n_1 \times c\}$ and $\{R(F),n_2,n_1 \times c\}$. Combine the two solutions to a two dimensional partitioning and find the total overflow.
2. Choose the factor pair which gives the minimum overflow.

□

## 4. Asymmetric Partitioning

In this section we describe the asymmetric algorithm for placing the horizontal and vertical lines so that a minimum overflow partitioning is achieved subject to the constraint on the number of pages in the primary area. As mentioned before, an asymmetric partitioning generalizes the symmetric one because it is less constrained in the way in which it partitions the attribute value space. Specifically, in the case of a temporal database we may partition the values of the surrogate into segments representing groups of surrogates and then perform a different horizontal partitioning for each such segment. The horizontal partitioning required for each segment may con-

tain a different number of cells due to the different activity rates of surrogate groups. For example in an inventory system some parts will be moved in and out of the warehouse at a much faster rate than others and therefore more transactions will be generated for these parts. This in turn will require allocating more horizontal lines to the segment which contains these parts.

The reason for partitioning the surrogate values into segments (rather than the time values) is that we expect that most queries will specify a surrogate value (or a small range) and try to access the records corresponding to this surrogate within a specified time range or obtain the whole time sequence for that surrogate. We would like to have an efficient data structure with respect to such queries while still supporting queries which specify a time range and then access transactions which refer to all or some subset of the surrogates.

An algorithm for solving this problem must perform the following:

a) Determine the number of vertical lines and their exact placement. This will partition the space into vertical segments.

b) For each vertical segment determined in a), find how many pages should be allocated to it so that the total number of pages allocated to all segments is equal to the constraint $K$. Assume that $k_i$ pages are allocated to the $i^{th}$ segment.

c) Find an optimal partitioning for the $i^{th}$ segment into $k_i$ pages. In terms of our previous algorithm, this requires placing $k_i - 1$ horizontal lines in the $i^{th}$ segment.

An algorithm for optimally solving this problem has to inspect a large number of choices for Step a). For the purpose of comparison with the symmetric algorithm, we devised two heuristic algorithms which we will call for reference purposes Algorithm 2 and Algorithm 3. In Algorithm 2, we fix the number of vertical segments to be as close as possible to $\sqrt{K}$. This has the effect of balancing between the number of lines placed in each dimension. In Algorithm 3 we looked at more choices for the number of vertical segments. In this case we inspected all the permissible factor pairs $(n_1, n_2)$ in P (as in Step 1 of Algorithm 1) and took the second component $n_2$ as a value for the number of vertical segments. This is the only difference between Algorithm 2 and Algorithm 3. In both cases, once the number of vertical lines was chosen, the exact placement of

these lines was determined by solving a one-dimensional partitioning problem on $C(F)$ the column sum vector of the matrix $F$ (as defined in the previous section). From now on, let us assume that step a) is completed and we have a fixed vertical partitioning $f$ with $n_2$ vertical segments.

Let us denote by $FL(i,j)$ the overflow incurred by optimally partitioning the $i^{th}$ segment into $j$ pages. We note that under the fixed partitioning $f$, finding the value of $FL(i,j)$ is a standard one dimensional dynamic program for every value of $i$ and $j$. Let $TF(j,l)$ denote the overflow incurred by the optimal partitioning of the first $j$ segments into a total of $l$ pages. We have the following recursive equation:

$$TF(j,l)=\underset{m}{Min}\{TF(j-1,m)+FL(j,l-m)\}.$$

This equation expresses the fact that the optimal partitioning of the first $j$ segments into $l$ pages is achieved by choosing some $m$ where $m<l$, and optimally partitioning the first $j-l$ segments into $m$ pages and then partitioning the $j^{th}$ segment into $l-m$ pages. The boundary condition is

$$TF(1,l)=FL(1,l) \quad \textit{for every } l.$$

A dynamic programming approach was used to find the value of $m$ which minimizes the above equation. As mentioned above, in Algorithm 2 only one value of $n_2$ was tested whereas in Algorithm 3, we computed $TF(n_2,K)$ for each $n_2$ in P and then chose as our optimal solution the vertical partitioning which produces the smallest value of $TF(n_2,K)$. The results of both algorithms and comparisons with the symmetric algorithm are presented in the next section.

## 5. Experimental Results

In this section we describe the computational experiments we conducted in order to compare the symmetric and the asymmetric partitioning of a database. The parameters of interest in these experiments were the number of possible attribute values, the constraint $K$ on the size of the database primary area and the storage utilization. For different combinations of these parameters, we were interested in comparing the total overflow generated by our three algorithms. Also we compared the quality of the solution generated by the three algorithms for a uniform input matrix $F$ as opposed to a matrix which has some clustering of values.

In Table 1 we compare the percentage of overflow generated by the three algorithms under four different storage utilization factors. Each entry in the table was computed as an average of four experiments. Each element of the matrix $F$ was generated as a uniform random variable in the range 1 to 1000. For each experiment, the size of the matrix $n$ , the number of pages $K$ , and the storage utilization factor was determined. The page capacity was computed from these numbers. As expected, Algorithm 3 always finds a solution with a smaller percentage of overflow. The amount of improvement between the best asymmetric algorithm and the symmetric one is shown in Figure 4. In this figure, we computed the ratio between the overflow reduction achieved by Algorithm 3 over the overflow generated by Algorithm 1 for different $n$ and storage utilization factors. In this graph we always chose $K=2n$ . As can be seen, the improvement becomes more significant with decreasing utilizaton factors. In general we observed reduction of overflow between 35 to 100 percent.

A similar set of experiments was also performed for an input matrix with clustered elements. This attempts to simulate situations in which some surrogates are more active than others or the existence of some time ranges in which more activity is expected in the file. We modeled this non-uniformity by partitioning our matrix into nine sub-matrices and then in each sub-matrix we generated uniform random numbers from different ranges as shown in Figure 5.

| | | |
|---|---|---|
| 10 | 1 | 1 |
| 1 | 1 | 10 |
| 5 | 15 | 1 |

Figure 5: A clustering scheme

Each number in this figure when multiplied by 100 gives the range used for generating random numbers in this region of the input matrix. The results of this set of experiments is shown in Table 2. In this case the asymmetric algorithm even performed better as it can adapt better to non uniformity in the input. The impact of storage utilization factor was less significant in this case as can be seen in the graph of Figure 6.

| $n$ | $K$ | Util.=0.80 | | | Util.=0.85 | | | Util.=0.90 | | | Util.=0.95 | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | Alg. 1 | Alg. 2 | Alg. 3 | Alg. 1 | Alg. 2 | Alg. 3 | Alg. 1 | Alg. 2 | Alg. 3 | Alg. 1 | Alg. 2 | Alg. 3 |
| 10 | 25 | 3.5 | 0.4 | 0.4 | 5.2 | 1.9 | 1.9 | 7.3 | 3.8 | 3.8 | 9.8 | 6.3 | 6.3 |
| | 40 | 4.4 | 1.3 | 0.3 | 6.3 | 2.6 | 1.2 | 8.3 | 4.3 | 2.8 | 10.5 | 6.6 | 4.9 |
| 20 | 40 | 0.6 | 0 | 0 | 1.3 | 0.1 | 0 | 2.6 | 1.1 | 0.6 | 4.3 | 2.6 | 1.9 |
| | 80 | 2.1 | 0.1 | 0 | 3.5 | 0.7 | 0.1 | 5.3 | 1.9 | 0.8 | 7.5 | 3.7 | 2.2 |
| 40 | 80 | 0.1 | 0 | 0 | 0.4 | 0 | 0 | 1.2 | 0.2 | 0.1 | 2.5 | 1.2 | 0.8 |
| | 160 | 0.6 | 0 | 0 | 1.4 | 0 | 0 | 2.8 | 0.1 | 0.1 | 4.6 | 1.9 | 1.1 |
| 80 | 160 | 0.2 | 0 | 0 | 0.1 | 0 | 0 | 0.3 | 0 | 0 | 1.3 | 0.4 | 0.2 |
| | 320 | 0.1 | 0 | 0 | 0.4 | 0 | 0 | 1.2 | 0.1 | 0 | 2.8 | 0.9 | 0.3 |
| 100 | 200 | 0.4 | 0 | 0 | 0.1 | 0 | 0 | 0.3 | 0 | 0 | 1.1 | 0.3 | 0.2 |
| | 500 | 0.1 | 0 | 0 | 0.5 | 0 | 0 | 1.3 | 0.1 | 0 | 2.8 | 0.8 | 0.3 |
| 200 | 200 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0.2 | 0 | 0 |
| | 500 | 0.6 | 0 | 0 | 0.1 | 0 | 0 | 0.1 | 0 | 0 | 0.7 | 0.1 | 0 |

**Table 1: Percentage overflow of the partitioning algorithms for uniform data**

| $n$ | $K$ | Util.=0.80 | | | Util.=0.85 | | | Util.=0.90 | | | Util.=0.95 | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | Alg. 1 | Alg. 2 | Alg. 3 | Alg. 1 | Alg. 2 | Alg. 3 | Alg. 1 | Alg. 2 | Alg. 3 | Alg. 1 | Alg. 2 | Alg. 3 |
| 10 | 25 | 28 | 7.5 | 7.5 | 31 | 10 | 10 | 34 | 12.6 | 12.6 | 35 | 14 | 14 |
| | 40 | 29 | 19 | 7.5 | 32 | 21 | 9.5 | 34 | 23 | 11 | 36 | 26 | 14 |
| 20 | 40 | 18 | 1.4 | 0.4 | 19 | 2.6 | 1.1 | 22 | 4.3 | 2.6 | 25 | 5 | 4 |
| | 80 | 28 | 7.1 | 0.5 | 30 | 8.8 | 1.5 | 33 | 10.2 | 3.1 | 35 | 12 | 5 |
| 40 | 80 | 17 | 0.1 | 0 | 19 | 0.5 | 0.1 | 20 | 1.7 | 0.7 | 23 | 3 | 2 |
| | 160 | 27 | 0.7 | 0 | 29 | 1.2 | 0.3 | 31 | 3.3 | 1.1 | 34 | 5 | 2 |
| 80 | 160 | 16.2 | 0 | 0 | 18 | 0.1 | 0 | 20 | 0.5 | 0.2 | 23 | 1.8 | 1 |
| | 320 | 27 | 0.1 | 0 | 29 | 0.6 | 0 | 31 | 1.7 | 0.2 | 34 | 3.5 | 1.3 |
| 100 | 200 | 16.6 | 0 | 0 | 18.8 | 0 | 0 | 20 | 0.2 | 0 | 23 | 1.3 | 0.7 |
| | 500 | 32 | 0.1 | 0 | 33 | 0.5 | 0 | 33 | 1.8 | 0.1 | 34 | 3.6 | 1.1 |
| 200 | 200 | 0.1 | 0 | 0 | 0.5 | 0 | 0 | 1.6 | 0 | 0 | 3.8 | 0.3 | 0 |
| | 500 | 26 | 0 | 0 | 28 | 0 | 0 | 29 | 0.1 | 0 | 32 | 0.8 | 0 |

Table 2: Percentage overflow of the partitioning algorithms for clustered data

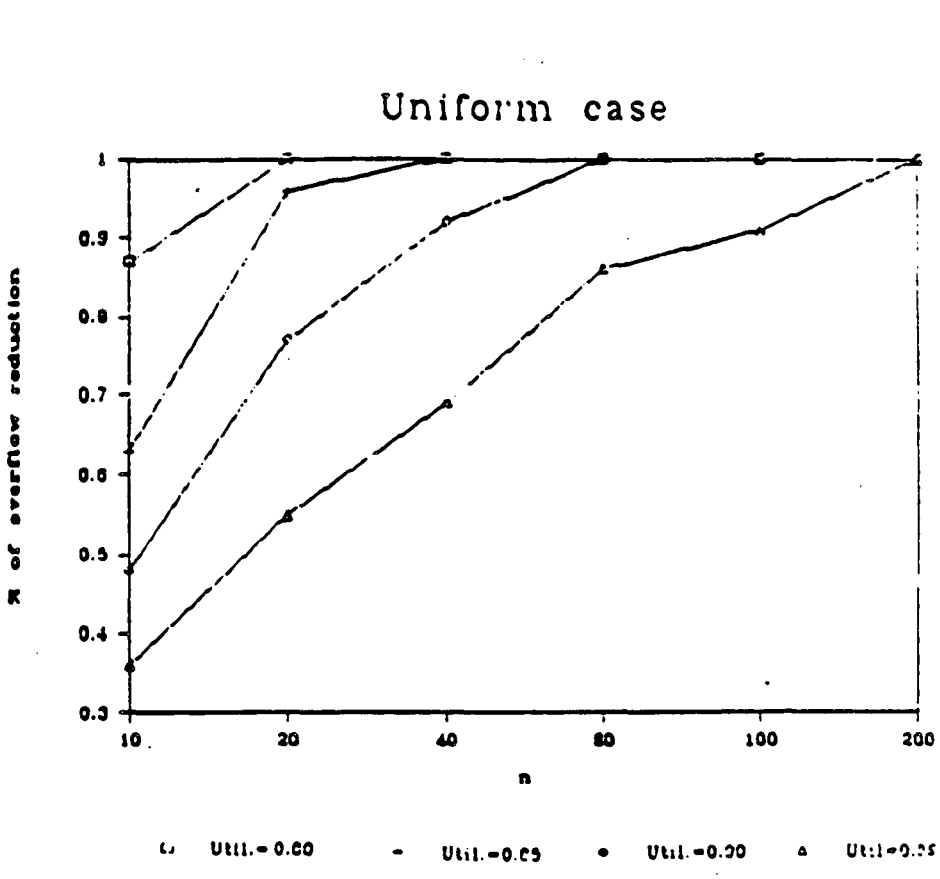



Figure 4

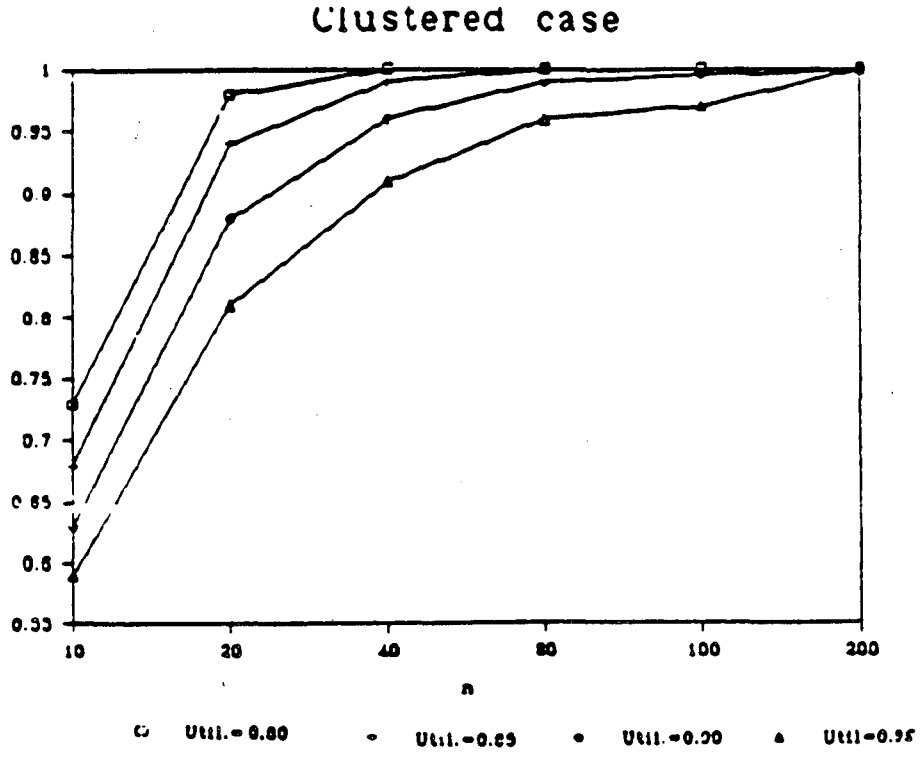



Figure 6

## 6. Conclusions

In this paper we examined two alternatives for multidimensional partitioning of temporal databases. It seems that the unique structure of these databases requires different data structures than simple grid files which perform very well for regular databases. The reason for this difference is that there is an inherent asymmetry of the time attribute with respect to the other attributes which requires special treatment. The structures proposed here are static in the sense that all the data or at least the distribution of the tuples over the attribute values must be known before a partitioning algorithm can be utilized. However, as shown in [ROTE86], static structures can be incorporated in a dynamic scheme for the purpose of conversion or periodical reorganization of a dynamic file to improve future performance. Also, although we did not deal with it in the paper, the asymmetric file can be used as a dynamic file where splits in the directory are made by adding a horizontal line in some vertical segment.

The main objective functions that we were trying to minimize were the total amount of overflow subject to a constraint on the number of pages $K$. The same techniques can be used for different objective functions. For example we can minimize $K$ subject to a zero overflow constraint or try to maximize storage utilization. Another important advantage of our scheme is that we can solve the dynamic program subject to constraints on where the partitioning lines must be placed. For example we can specify that surrogates $s_1$, $s_2$ and $s_3$ must belong to the same segment. The reason for this constraint is that a very common query type specifies all these three surrogates. In other words we can adapt our partitioning scheme to the query pattern as well as the data.

## 7. Acknowledgements

The authors wish to thank Dr Shoshani for his suggestions and help in preparing this paper.

# REFERENCES


AHN86 Ahn I., "Towards an Implementation of Database Management Systems with Temporal Support", **Proceedings of the International Conference on Data Engineering,** 1986, pp. 374-381.

ARIA84 Ariav G., Beller A., and Morgan H., "A Temporal Data Model", **Technical Report, New York University,** December, 1984.

BOLO82 Bolour A., Anderson T.L., Dekeyser L.J., and Wong H.K.T., "The Role of Time in Information Processing: A Survey", **IACM-SIGMOD Record,** 12, 3, 1982, pp. 27-50.

CLIFF85 11 Clifford J., and Tansel A., "On an Algebra for Historical Relational Databases: Two Views", **Proceedings of the ACM SIGMOD International Conference on Management of Data,** May 1985, pp. 247-265.

HORO78 Horowitz E., Sahni S., "Fundamentals of Computer Algorithms", **Computer Science Press, Inc.,** 1978.

LUM84 Lum V., Dadam P., Erbe R., Guenauer J., Pistor P., Walch G., Werner H., and Woodfill J., "Designing Dbms Support for the Temporal Dimension", **Proceedings of the ACM SIGMOD International Conference on Management of Data,** June 1984, pp. 115-130.

MERR78 Merret T.H.,"Multidimensional Paging for Efficient Database Querying", **ICMOD 1978** pp. 277-290.

MERR82 Merret T. H.,Otoo E.J.,"Dynamic Multipaging: A Storage Structure for Large Shared **Data Banks", Improving Database Usability and Responsivenes,** P. Scheurmann ed. (Jerusalem , June 1982), Academic Press New York, pp. 237-256.

MERR84 Merret T.H., "Relational Information Systems" **Reston Publishing Company,** 1984.

NIEV84 Nievergelt J.,Hinterberger H., and Sevcik K.C., "The Grid File: An Adaptable, Symmetric Multikey File Structure" **ACM TODS,** Vol. 9, 1 (March 1984), pp. 38-71.

OZKA85 Ozkarahan E.A. and Ouksel M., "Dynamic and Order Preserving Data Partitioning for Database Machines", **VLDB,** Stockholm, Sweden, 1985, pp. 358-368.

OZKA85a Ozkarahan E.A., "Database Machines and Database Management", **Prentice-Hall Inc.,** 1985.

REGN85 Regnier M.,"Analysis of Grid File Algorithms", **BIT,** Vol 25 (1985), pp. 335-337.

RUBE85 Rubenstein B., "Indexes for Time-Ordered Data" **Technical report** Computer Science Dept., University of California, Berkeley, 1985.

SNOD84 Snodgrass R., "The Temporal Query Language TQuel", **Proceedings of the Third ACM SIGMOD Symposium on Principles of Database Systems (PODS)**, Waterloo, Canada, April 1984, pp. 204-213.

SNOD85 Snodgrass R., and Ahn I., "A Taxamony of Time in Databases", **Proceedings of the ACM SIGMOD International Conference on Management of Data**, May 1985, pp. 236-246.

ROTE86 Rotem D. and Segev A., "Algorithms for Multidimensional File Partitioning", **Submitted for publication.**

SHOS86 Shoshani A. and Kawagoe K., "Temporal Data Management", **VLDB**, Kyoto, Japan, 1986. To appear.

This report was done with support from the Department of Energy. Any conclusions or opinions expressed in this report represent solely those of the author(s) and not necessarily those of The Regents of the University of California, the Lawrence Berkeley Laboratory or the Department of Energy.

Reference to a company or product name does not imply approval or recommendation of the product by the University of California or the U.S. Department of Energy to the exclusion of others that may be suitable.

*LAWRENCE BERKELEY LABORATORY*
*TECHNICAL INFORMATION DEPARTMENT*
*UNIVERSITY OF CALIFORNIA*
*BERKELEY, CALIFORNIA 94720*