ELSEVIER

Data & Knowledge Engineering 18 (1996) 147–165

DATA & KNOWLEDGE ENGINEERING

Indexing temporal data using existing B$^+$-trees

Cheng Hian Goh[†], Hongjun Lu*, Beng-Chin Ooi, Kian-Lee Tan

Department of Information Systems and Computer Science, National University of Singapore, 10 Kent Ridge Crescent, Singapore 0511, Singapore

Received 22 August 1994; revised 24 March 1995; accepted 4 September 1995

Abstract

Research in temporal databases has largely focused on extensions of existing data models for the proper handling of temporal information. One approach is to store temporal data on existing DBMS and build some new indexes to provide support for the efficient retrieval of temporal data. This paper describes mapping strategies to linearize the data such that existing B$^+$-trees can be used directly. With such an implementation, a temporal relation is mapped to points in a multi-dimensional space, with each time interval being translated to a two-dimensional coordinate, and a temporal selection operation is constructed as a spatial search operation. The proposed approach has two advantages. First, mapping a temporal relation to a multi-dimensional space provides a uniform framework for dealing with temporal queries involving transaction and valid time, as well as other non-temporal attributes. Second, linearization of the multi-dimensional search space allows classical indexing methods (such as the B$^+$-tree) to be used; this means that index support for temporal selection can be accomplished without modification to the underlying storage components of the DBMS. Both analytical and simulation study show that the proposed indexing scheme is more efficient than the time index in both its disk utilization and access time.

Keywords: Temporal database; Indexing techniques; B$^+$-tree; Spatial selection; Relational database

1. Introduction

Research in temporal databases has largely focused on extensions of existing data models for the proper handling of temporal information [1, 3, 5, 6, 18, 20–22]. More recently, many researchers have expressed an interest in the viable implementation of these temporal operators. To this end, a number of specialized storage structures or indexes have been proposed to provide support for the efficient retrieval of temporal data [4, 10, 12, 15, 17]. However, the implementation of these indexing methods entails substantial changes to the

[†] Author's current address: Sloan School of Management, Massachusetts Institute of Technology, Room E53-308, 50 Memorial Drive, Cambridge, MA 02139. Internet: chgoh@athena.mit.edu

* Corresponding author. Email: luh@iscs.nus.sg

0169-023X/96/$15.00 © 1996 Elsevier Science B.V. All rights reserved
SSDI: 0169-023X(95)00034-8

storage component of a DBMS. For most practical purposes, such changes may be too expensive to be viable.

In this paper, we propose a simple but efficient approach to temporal indexing. Our objective is three-fold. First, we would like to propose a temporal indexing scheme which provides a uniform approach for handling the different notions of time, as well as non-temporal attributes. Second, we would like the implementation to be non-intrusive, so that it can be useful in the context of today's technology. Finally, and perhaps most importantly, the performance of our indexing scheme should be more efficient than existing indexing methods.

The rest of this paper is organized as follows. Section 2 provides the background to our study. Section 3 reviews briefly the various proposals for temporal indexing. The *time index* [4] is presented in a little more detail here to facilitate its comparison to the method proposed in this paper. In Section 4, we illustrate how different types of temporal selections can be mapped to search operations in a multi-dimensional search space. Having reduced the temporal selection operation to a spatial search operation, we demonstrate in Section 5 that points in this search space can be indexed using classical indexing schemes by defining a linear order on these points. Section 6 analyses the performance of a B^+-tree implementation using the proposed approach and the time index. In Section 7, we illustrate how queries that involve different notions of time can be supported easily using the proposed index. Finally, we present the conclusion in Section 8 and provide a glimpse of extensions to this work.

2. Background

Following [5] and others, we represent the time dimension using both discrete time points and time intervals. A *time interval*, denoted by $[a, b]$ is defined to be a set of consecutive equidistant time instants (points), where a is the first time instant and b is the last time instant of the interval. The *time dimension* is represented as a time interval $[0, now]$, where 0 represents the starting time of our application and *now* refers to the current time which is continuously increasing.

In [2], three notions of time were accepted to have utility in real world databases:

- *transaction time* refers to the time when information is stored in the database;
- *valid time* corresponds to the time when the information recorded in the database is valid; and
- *user-defined time* corresponds to additional temporal information which are present in user-defined attributes.

For a large part of this paper (except Section 7), we will adopt a simple view of a temporal database so that we can concentrate on the essential issues related to indexing a temporal database. Throughout, unless otherwise stated, we will use the valid time in our discussion. However, since both the transaction and valid time are represented as intervals, we should bear in mind that the discussion applies also to transaction time. A tuple μ corresponding to a relation in the temporal database has associated with it, a valid time interval $[v_s, v_e]$, denoted by $\mathcal{V}(\mu)$. We shall use the following example continually in the rest of the paper for exemplifying our discussion.

tuple	pid	entry_pt	$\mathcal{V}(\mu)$
t1	p1	NY	[0,3]
t2	p1	LA	[4,*now*]
t4	p2	SFO	[0,5]
t7	p3	LA	[0,7]
t8	p3	SFO	[8,9]
t10	p4	NY	[2,3]
t11	p4	LA	[8,*now*]
t12	p5	LA	[10,*now*]
t13	p6	NY	[12,*now*]
t14	p7	NY	[11,*now*]

Fig. 1. The `arrival` relation in Example 1.

Example 1. Consider an FBI database which keeps record of a list of persons and their visits to the United States. An `arrival` relation for this database is shown in Fig. 1. Every person in this database is assigned a unique `pid`. The attribute `entry_pt` captures the information on the entry point used to gain entrance to the United States. Arrival and departure time is normalized against some reference point (day 0) and hence the duration of each person's visit can be represented as a time interval in the time dimension $[0, now]$. Each tuple is assigned a unique tuple id to facilitate references to it.

3. Approaches to temporal indexing

Obviously, the full benefits of temporal databases can only be realized if data retrieval based on the different notions of time can be supported efficiently. In this section, we provide a brief survey of the various temporal indexing approaches.

In [13], Lum et al. suggested that past versions of an object can be linked or indexed separately to facilitate their retrieval. However, this means that locating valid versions in a given time interval is cumbersome. Rotem and Segev [15] proposed that the time dimension can be viewed as one of the dimensions in a multi-dimensional space and hence temporal data can be organized using a multi-dimensional partition file. A severe limitation of their approach lies in their inability to handle time intervals effectively. Both Lomet and Salzberg [12] and Kolovson and Stonebraker [10] studied the problem with the assumption that historical data is stored separately from current data in an optical disk. Consequently, their model is appropriate only in the context of a rollback database. The *AP-tree* proposed in Segev and Gunadhi [17] is primarily designed to support the optimization of *event-joins* and does not support temporal selections effectively.

More recently, Elmasri et al. [4] proposed the *time index* which is aimed at providing efficient access to temporal data in some valid time interval. The idea behind the time index is to maintain a linearly ordered set of *indexing points* on the time dimension $[0, now]$. This set BP is formed from *now* and all the time points corresponding to (a) the valid start time of some tuple, and (b) the time point immediately after the valid end time of some tuple. Since these indexing points can be linearly ordered, a regular B^+-tree is used to index these. Each

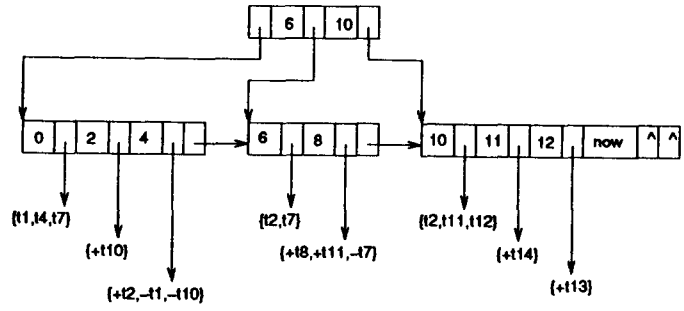


Fig. 2. The time index constructed from the most current snapshot of the `arrival` relation.

leaf node entry of this B^+-tree contains a collection of entries of the form (t_i, B_i), where B_i is a pointer to a bucket containing pointers to tuples which are valid at time t_i. To reduce the number of pointers which are stored in the buckets, only the leading bucket in each leaf node is retained. All subsequent buckets in the leaf node record only the incremental changes. Fig. 2 depicts the time index constructed using the most current snapshot of the `arrival` relation in Example 1.

The time index can be used to process *historical* queries reasonably efficiently. For instance, the query "Find all persons who were in the United States from day 4 to day 6" can be answered by locating indexing point 4, and reconstructing the list of valid tuples from the leading bucket and subsequent entries right up to indexing point 6. This represents a vast improvement over scanning the entire database sequentially. However, like its predecessors, index support is provided for only a single notion of time (in this case, valid time) and it is not clear how this can be extended to support temporal queries which necessarily involves both transaction and valid time.

In [4], Elmasri et al. suggested that their time index can also be appended to regular indexes to facilitate processing of historical queries involving other non-temporal search conditions. For example, if queries such as "Find all persons who entered the United States via LA and remained from day 4 to day 6" is expected on a regular basis, such queries may be supported by attaching a time index structure to each leaf entry of a B^+-tree constructed for the attribute `entry_pt`. Answering the above query will then involve traversing the first B^+-tree to first identify the leaf entry corresponding to attribute value "LA", followed by an interval search on the time index found there. However, this approach may not be scalable at all since the number of time indexes will certainly grow to be exorbitantly large in any non-trivial database.

The major drawback of the implementation of any of the above indexing schemes is that it involves substantial changes to the storage components of existing DBMSs. This means that existing databases brimming with temporal data cannot benefit from these proposals unless one is willing to undertake the formidable task of revamping the underlying DBMS.

4. Temporal selection as spatial search

In this section, we introduce a mapping strategy for temporal indexing which, to some extent, is based on spatial indexing techniques. Although spatial indexing has been very well researched [16, 14], it is widely recognized that spatial indexes have not been designed to

efficiently support temporal data [4]. This is because, unlike spatial objects which have extents, temporal data are lines parallel to the time axis and have zero extent on the text-attribute dimension. Indexes such as R-trees [7] and R^+-tree [19], designed to handle non-zero sized multi-dimensional spatial objects, cannot therefore be readily used for temporal indexing.

We overcome the above problem via a transformation which maps a given time interval (whether valid time or transaction time) to a two-dimensional coordinate [9, 11]. Under this mapping, any temporal selection (whether involving valid time, transaction time, or non-temporal attributes) can be mapped to a spatial search operation in a multi-dimensional space. We call such a transformation on time intervals *interval-spatial transformation* (IST).

Definition 1. Let $\mathcal{I}$ be the set of all time intervals $[a, b]$ in the time dimension $[0, now]$, and $\mathcal{P}$ be the set of discrete time points in the time dimension. The *interval-spatial transform*, denoted by $\mathcal{T}$, is a function from $\mathcal{I}$ to $\mathcal{P}^2$, such that $\mathcal{T}([a, b]) = (a, b - a)$.

The function $\mathcal{T}$ can be applied to any time interval regardless of its semantics (i.e. whether it is a transaction time interval or a valid time interval). The utility of this transformation is exemplified below.

Example 2. Consider the snapshot of the `arrival` relation at *now*. These tuples can be mapped to discrete points in a two-dimensional space by defining a mapping function $\mathcal{M}$ on this relation, such that for each tuple μ, $\mathcal{M}(\mu) = \mathcal{T}(\mathcal{V}(\mu))$. For example, $\mathcal{M}(t10) = (2, 1)$ since the valid time interval corresponding to tuple t10 is $[2, 3]$. Fig. 3 shows the result after applying the mapping $\mathcal{M}$ to the most recent snapshot of `arrival`.

Observe that Fig. 3 provides a highly visual representation of three important parameters associated with the valid time interval of each tuple.

(1) any tuple with a valid start time $v_s = a$ must fall on the line $x = a$;

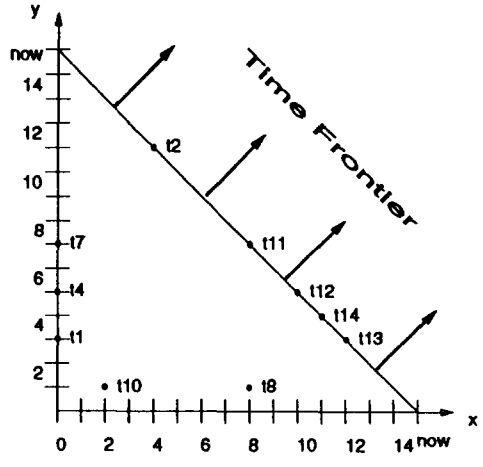


Fig. 3. Mapping the most current snapshot of the `arrival` relation to discrete points in a two-dimensional space based on the valid time interval of each tuple.

(2) any tuple with a valid end time $v_e = b$ must fall on the line $x + y = b$; and
(3) any tuple with a total valid time duration of c must fall on the line $y = c$.

With the above observations, it is not difficult to understand how temporal queries on the relation can be mapped to spatial search operations in this two-dimensional space. Consider the following queries:

- List all persons who entered the country on or before day t_a.
- List all persons who left the country on or after day t_b.
- List all persons who remained in the country for a total duration of t_c or less days.

The answers to each of these queries can be found by retrieving all points which fall in the regions shown in Fig. 4(a), (b) and (c), respectively.

Since we assumed that the time dimension is discrete, selections such as "List all persons who entered the country before t_a" can be easily transformed to "List all persons who entered the country *on or* before $t_a - 1$". Moreover, conjunctions of selection criteria refers naturally to the intersection of those regions corresponding to the individual selection criterion. For

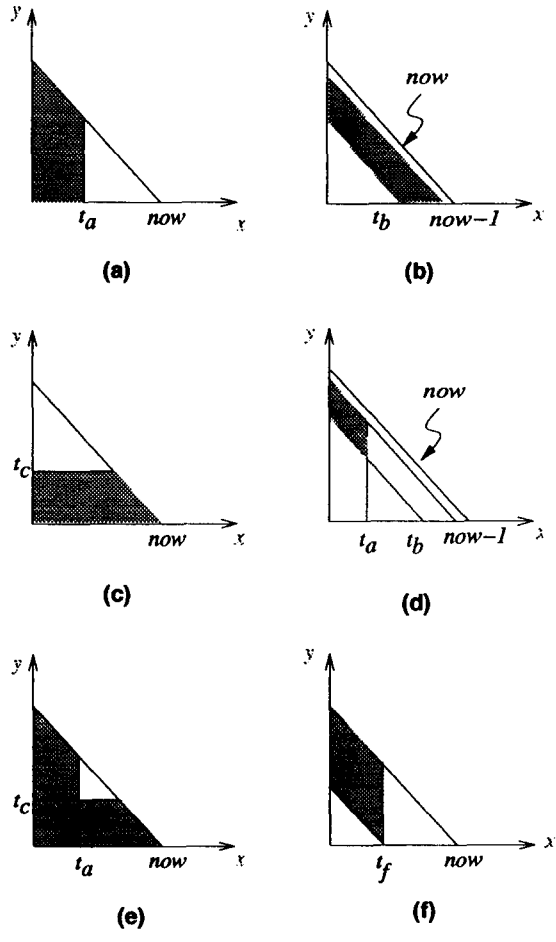


Fig. 4. Temporal selections based on valid time and their corresponding search spaces.

instance, the query "List all persons who entered the country on or before t_a and left on or after t_b", shown in Fig. 4(d), is simply the intersection of the two regions in Fig. 4(a) and (b). Similarly, disjunctions of selection criteria can be modeled as the union of respective regions. For example, the query "List all persons who entered on or before t_a or remained for t_c days or less" can be answered by retrieving all points in the region shown in Fig. 4(e). Finally, a degenerate instance of such queries corresponds to selection on time points, such as "Find all persons who were in the United States on t_f" corresponds to the search space shown in Fig. 4(f). □

The above transformation can be extended to temporal selections involving non-temporal attributes. Suppose Q is a query on some temporal relation r having attributes $A_1, \ldots, A_m$, and suppose tuples are to be selected based on some restrictions of the values of attributes $A_1, \ldots, A_n$, where $n \leq m$, and their corresponding valid times. We can define a function $\mathcal{M}$ on r which maps to an $(n+2)$-dimensional space. Hence, if μ is a tuple in r,

$$\mathcal{M}(\mu) = (x, y, \alpha_1, \ldots, \alpha_n)$$

where $(x, y) = \mathcal{T}(\mathcal{V}(\mu))$ and $\alpha_i = \mu[A_i]$ for $i = 1, \ldots, n$.

Example 3. Consider once again the most current snapshot of the `arrival` relation in Example 1. Suppose we are interested in defining an efficient access path for queries of the form "Find all persons who entered via LA before day 6, and who remained until after day 10." In this query, the selection criteria are (i) the valid time interval for each tuple, and (ii) the entry point, which is the value corresponding to the attribute `entry_pt`. The mapping $\mathcal{M}$ on this relation can thus be defined as follows: suppose μ is a tuple of `arrival`, then $\mathcal{M}(\mu) = (x, y, e)$, where $(x, y) = \mathcal{T}(\mathcal{V}(\mu))$ and $e = \mu[\texttt{entry\_pt}]$. For instance, tuple t10 will be mapped to the point (2, 1, "NY"). Fig. 5 shows the result of applying $\mathcal{M}$ on the most current snapshot of `arrival`. The answer to the earlier query can be found by identifying those points which now fall in the shaded region as shown. As is evident from Fig. 5, the only tuple which satisfies the selection criteria is t2.

5. Indexing the spatial representation

In the previous section, we have shown that temporal retrieval (based on valid time and/or non-temporal attributes) can be transformed to spatial search operations in a multi-dimensional space constructed from the target relation. Clearly, a viable solution for the spatial indexing problem (e.g. the *grid file* [9], the *LSD tree* [8] and other data structures for handling point data [16]) could be adapted to provide the needed indexing support for temporal data. Notwithstanding this, we will, in the remainder of this paper, explore an alternative approach to indexing which exploits the peculiar characteristics of temporal search spaces. Specifically, we demonstrate that, depending on the types of temporal queries most frequently executed, points in a multi-dimensional space can be linearized using a number of orderings. Consequently, conventional indexing mechanisms (such as the B^+-tree) can now be employed for indexing these points. For the sake of simplicity, we shall confine our discussion to the

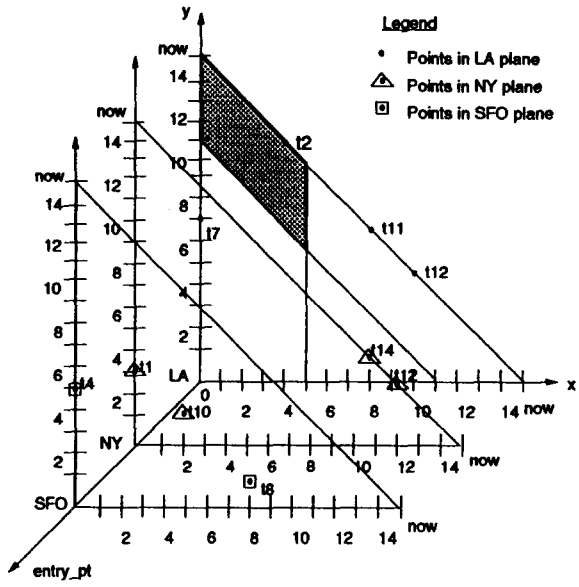


Fig. 5. A three-dimensional representation of the most current snapshot of `arrival`. The shaded region represents the search space corresponding to the query "Find all persons who entered via LA before day 6 and who remained until after day 10".

indexing of points in a two-dimensional space. The proposed approach however, can be extended for indexing points in a higher dimension.

Consider a two-dimensional space similar to the ones shown in Fig. 3. Points in this two-dimensional space can be mapped to a one-dimensional space by defining a linear order on them. While infinitely many linear orders can probably be identified, only a handful are likely to be useful. Recall that $\mathcal{P} = \{0, \ldots, now\}$ (i.e. the set of time instants in the time dimension); we shall introduce three linear orders for those points in the two-dimensional space $\mathcal{P}^2$. We refer to these as the *D(iagonal)-order*, the *V(ertical)-order* and the *H(orizontal)-order*, and denote them by $<_D$, $<_V$ and $<_H$, respectively.

Definition 2. Let $P_1(x_1, y_1)$ and $P_2(x_2, y_2)$ be two distinct points in $\mathcal{P}^2$. We say that

(1) $P_1 <_D P_2$ iff
  (a) $(x_1 + y_1) < (x_2 + y_2)$; or
  (b) $(x_1 + y_1) = (x_2 + y_2)$ and $x_1 < x_2$;
(2) $P_1 <_V P_s$ iff
  (a) $x_2 + y_2 = now$ and $x_1 < x_2$; or
  (b) $x_1 + y_1 \neq now$ and $x_2 + y_2 \neq now$ and $x_1 < x_2$; or
  (c) $x_1 + y_1 \neq now$ and $x_2 + y_2 \neq now$ and $x_1 = x_2$, and $y_1 < y_2$;
(3) $P_1 <_H P_2$ iff
  (a) $x_2 + y_2 = now$ and $y_1 < y_2$; or
  (b) $x_1 + y_1 \neq now$ and $x_2 + y_2 \neq now$ and $y_1 < y_2$; or
  (c) $x_1 + y_1 \neq now$ and $x_2 + y_2 \neq now$ and $y_1 = y_2$, and $x_1 < x_2$;

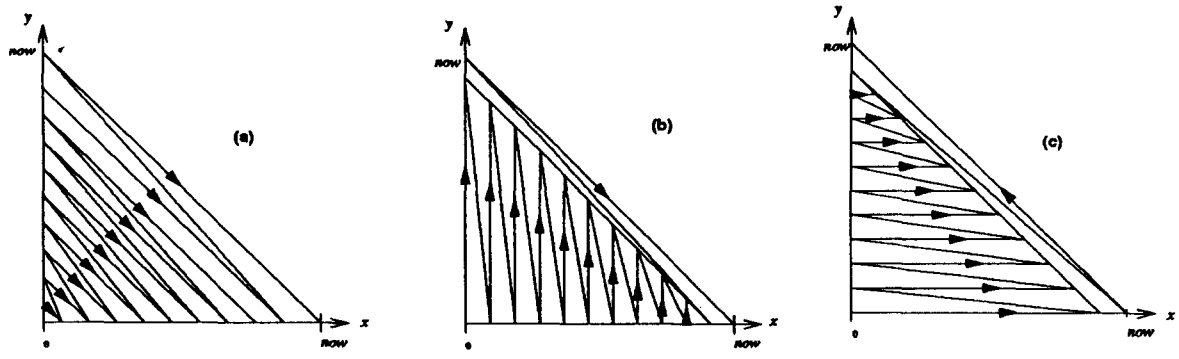


Fig. 6. Three alternate orderings (D, V and H) for points in the two-dimensional search space.

Fig. 6 provides a graphic representation of the three linear orders defined above. Notice that in adopting any one of these linear orders, points in $\mathscr{P}^2$ can now be organized using an index such as the B^+-tree. Consequently, spatial search operations on this two-dimensional space can be translated to range search operations on the linear space defined by the ordering relation.

Example 4. Consider the spatial representation of the most current snapshot of `arrival` depicted in Fig. 3. We can index these points using conventional indexing schemes by adopting any linear order defined on $\mathscr{P}^2$. For instance, if we order these points using the D-order, then we may say that $t1(0,3) <_D t10(2,1)$ since both points fall on the line $x + y = 3$ and $0 < 2$. Also, $t8(8,1) <_D t2(4,11)$ since $8 + 1 = 9 < 4 + 11 = now$. Fig. 7 depicts a B^+-tree which is used for indexing the spatial representation found in Fig. 3, while adopting the D-order.

The motivation behind the choice of linear orders is best illustrated with an example. Consider the query "Find all persons who left the United States on or after day 5." The search space is similar to that shown in Fig. 4(b), where t_b is now 5. If *now* is day 15, the points in

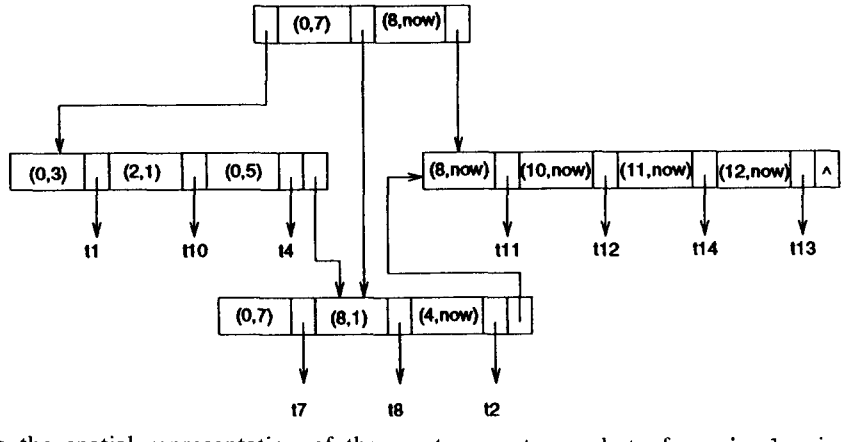
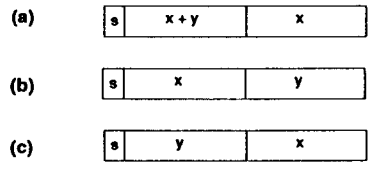


Fig. 7. Organizing the spatial representation of the most current snapshot of `arrival` using a B^+-tree and linearizing using the D-order.

this search space can be found by traversing the D-order B^+-tree and retrieving all points in the interval [(0,5), (14,0)]. The answer is clearly persons corresponding to tuples t4, t7 and t8. However, not all temporal queries can be mapped to a simple range search; it may be necessary for the spatial search to be decomposed into a number of interval queries. For instance, the query "Find those persons who were in the United States on day 11" would be decomposed into the following interval searches: [(0,11), (11,0)], [(0,12), (11,1)], [(0,13), (11,2)], [(0,14), (11,3)], and [(0,15), (11,4)] (assuming that *now* is 15).

A careful examination of Fig. 4 reveals that different classes of temporal queries give rise to spatial search spaces of different forms. Intuitively, temporal queries with search spaces similar to Fig. 4(b) are best supported by a D-order B^+-tree. Similarly, queries similar to Fig. 4(a) are best supported by a V-order B^+-tree, and those of Fig. 4(c), by a H-order B^+-tree. This suggests that different indexes (constructed using different ordering relations) may be used to support the various types of queries. Where multiple indexes exist, the selection of a most appropriate index structure constitutes an access path optimization problem.

From a practitioner's perspective, the most important feature of our approach is perhaps the ease with which this indexing scheme can be implemented using existing DBMSs. Unlike existing temporal indexes, the implementation of our indexing approach does not require any changes to the storage component of existing DBMS. The different orderings for points in $\mathcal{P}^2$ (and hence different access paths) can be effected by altering the manner in which each coordinate (x, y) is mapped to a linear bit string. Suppose we are to construct a D-order B^+-tree: if $x + y$ can be represented using n bits, then we can represent each coordinate with a bit-string of length $2n + 1$; the lower n bits is used for representing the x-value, with the next n bits representing the $x + y$-value; finally, if the y-value happens to be *now*, the leading bit s is set to one (otherwise it is 0), and the $x + y$-field is set to 0. Fig. 8 illustrates the bit-string



tuple	coordinate (x, y)	D-order rep.	V-order rep.	H-order rep.
t1	(0,3)	030	003	030
t2	(4,*now*)	104	140	104
t4	(0,5)	050	005	050
t7	(0,7)	070	007	070
t8	(8,1)	098	081	018
t10	(2,1)	032	021	012
t11	(8,*now*)	108	180	108
t12	(10,*now*)	10A	1A0	10A
t13	(12,*now*)	10C	1C0	10C
t14	(11,*now*)	10B	1B0	10B

Fig. 8. Effecting the D-, V- and H-ordering via a bit-string representation of a co-ordinate point (x, y). The hexadecimal equivalents of the most current snapshot of `arrival` is shown while assuming that $n = 4$.

representations for the D-, V- and H-order diagrammatically. The accompanying table also shows the key values for tuples belonging to the most recent snapshot of `arrival`.

6. Analytical evaluation

In this section, we study the performance of the proposed scheme by comparing both the storage requirement and query performance of a B^+-tree implementation (based on the proposed method) with the time index proposed by Elmasri et al. [4].

We begin by stating some of the assumptions about the system parameters:

- block size, $B = 4$ Kbytes;
- pointer size, $p = 32$ bytes;
- integer size (corresponding to time points), $t = 8$ bytes.

With these values, the *order* of our B^+-tree, m_0, can be determined directly:

$$2m_0(p + 2t) + p = B \Rightarrow m_0 = \left\lfloor \frac{B - p}{2(p + 2t)} \right\rfloor \approx 42$$

(This means that each node of this B^+-tree has at least 42 and at most 84 entries.) The order of the B^+-tree for the time index, m_e is slightly larger since the key value is merely one indexing point (one time point) as opposed to the ordered pair in ours. Thus, we have

$$2m_e(p + t) + p = B \Rightarrow m_e \approx 50$$

We also make the assumption that (in the case of the time index) all bucket entries corresponding to the same leaf node are clustered together whenever possible. This cuts down the number of disk pages needed for bucket storage. The number of bucket entries which can fit into a single disk page is approximately $2m_e$ or 100.

The remainder of the section addresses two performance measures: (i) storage requirements of the indexing scheme, and (ii) the performance of the two indexing methods under a variety of temporal queries, as measured by the number of disk accesses needed. For convenience, we assume that the target relation is none other than the `arrival` relation presented in Example 1. We also assume that the arrival of persons is a Poisson process, and hence inter-arrival time is exponentially distributed with mean $1/\lambda$. The duration of each visit is assumed to be uniformly distributed over the interval $[0, 2\mu]$.

6.1. Storage requirements

We shall first derive an analytical model for the storage requirements of Elmasri's time index. Let S_e be the total number of disk blocks which are consumed for a relation having N tuples. Clearly, $S_e = S_e^i + S_e^b$, where S_e^i is the number of disk blocks used by the time index B^+-tree structure, and S_e^b refers to the number of disk blocks needed to contain all the bucket entries.

Obviously, the size of the time index B^+-tree depends on the number of distinct indexing points in BP (the set of base points which are used in constructing the time index). Suppose X

is a Poisson random variable which represents the number of arrivals in unit time (in this example, one day). $|BP|$ can be approximated by

$$2N \times \text{Prob}(X > 0) = 2N(1 - (1 - e^{-\lambda})) = 2N e^{-\lambda}$$

Assuming that the nodes in the B$^+$-trees is ln 2 full on average [23], the number of leaf nodes is given by

$$\left\lceil \frac{2N e^{-\lambda}}{2m_e \ln 2} \right\rceil = \left\lceil \frac{N e^{-\lambda}}{m_e \ln 2} \right\rceil$$

If we account for the non-leaf nodes as well, then

$$S_e^i = \left\lceil \frac{N e^{-\lambda}}{m_e \ln 2} \right\rceil \left(1 + \frac{1}{2m_e \ln 2} + \cdots\right) \approx \frac{N e^{-\lambda}}{m_e \ln 2} \times \left(\frac{2m_e \ln 2}{2m_e \ln 2 - 1}\right)$$

Each leaf node of the time index B$^+$-tree makes use of at least one disk block for its bucket entries. We estimate the number of entries in the leading bucket of each leaf node by the expected number of arrivals in a period μ, which yields the value $\mu\lambda$. The number of incremental entries for each leaf node is merely (expected number of time points in each leaf mode) × (expected number of arrivals at any time point) = $(2m_e \ln 2)\lambda$. By assuming that the bucket pages are packed as tightly as possible, we have

$$S_e^b = \left\lceil \frac{N e^{-\lambda}}{m_e \ln 2} \right\rceil \times \left\lceil \frac{\mu\lambda + 2m_e \lambda \ln 2}{2m_e} \right\rceil$$

Consequently,

$$S_e = \left\lceil \frac{N e^{-\lambda}}{m_e \ln 2} \right\rceil \times \left[\left(\frac{2m_e \ln 2}{2m_e \ln 2 - 1}\right) + \left\lceil \frac{\mu\lambda + 2m_e \lambda \ln 2}{2m_e} \right\rceil \right]$$

The above expression suggests that even for a fixed N, the storage requirements of the time index varies with the characteristics of the temporal data. We may interpret this intuitively as follows: if the inter-arrival time is small, multiple arrivals at the same time become likely and hence the number of distinct indexing points is reduced. This leads to a smaller B$^+$-tree and hence lower disk utilization. If however, the (valid time) duration of each tuple is large compared to the inter-arrival time, the same tuple is potentially duplicated across many leading buckets and this leads to large storage overheads. Fig. 9 depicts this graphically by plotting the total number of disk blocks used against $1/\lambda$ and the ratio $\mu\lambda$.

Unlike the time index, the B$^+$-tree constructed on the mapped temporal data is not dependent on the characteristics of the temporal data. This is due to the fact that each tuple is represented only once, and hence the number of indexing values is bounded by N (the total number of tuples in the temporal relation). Suppose S_0 denotes the number of disk blocks required for constructing a B$^+$-tree under our scheme. The approximation of this value is identical to the analysis for classical B$^+$-tree:

$$S_0 = \left\lceil \frac{N}{2m_0 \ln 2} \right\rceil \left(1 + \frac{1}{2m_0 \ln 2} + \cdots\right) \approx \left\lceil \frac{N}{2m_0 \ln 2} \right\rceil \times \left(\frac{2m_0 \ln 2}{2m_0 \ln 2 - 1}\right)$$

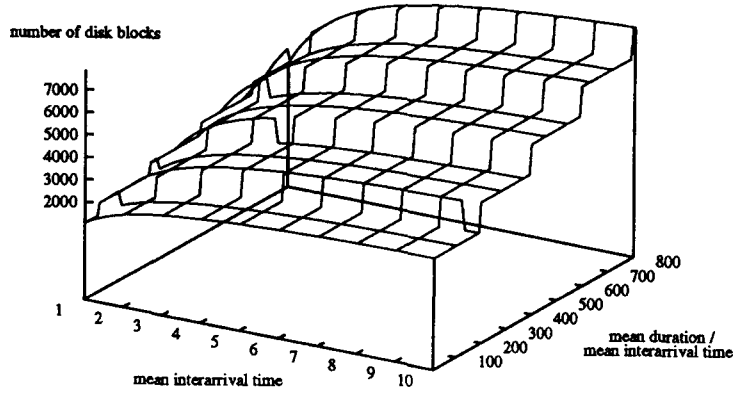


Fig. 9. Three-dimensional plot depicting the relationship between disk utilization with inter-arrival time (mean = $1/\lambda$) and valid time duration (mean = μ) for Elmasri's time index.

This expression is independent of $1/\lambda$ and μ.

Fig. 10 shows the results obtained from a simulation study in which N is kept constant at 50 000, and the inter-arrival time $1/\lambda$ and valid time duration μ is varied. The results confirm our analysis: disk utilization for a B^+-tree constructed for temporal indexing using our scheme remains practically constant with no regard for μ or λ; the disk utilization for the time index, on the other hand, fluctuates as a function of both of these parameters.

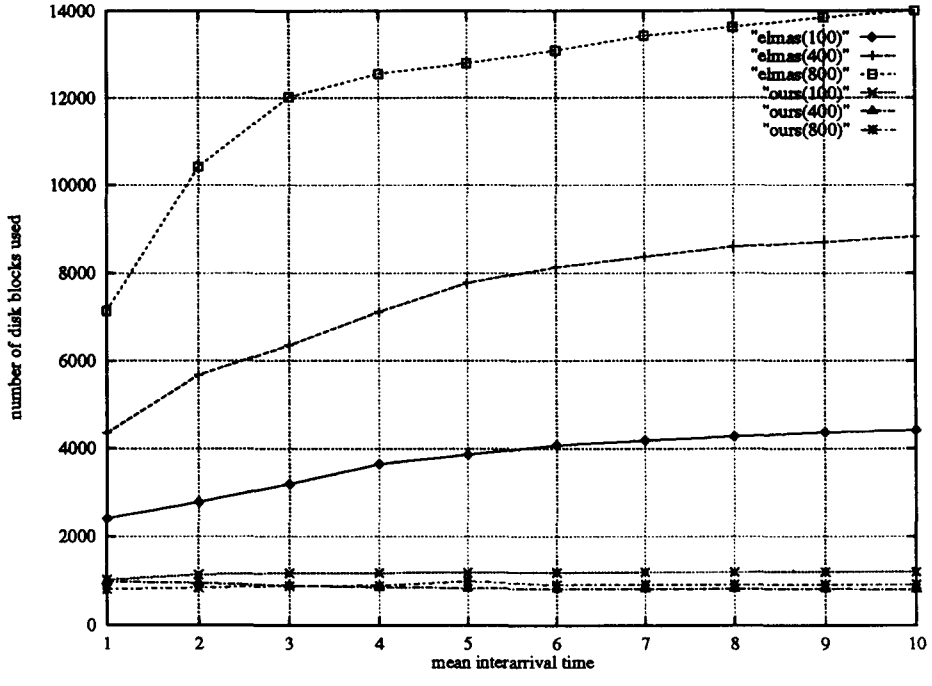


Fig. 10. Experimental result depicting how storage utilization varies with inter-arrival time and valid time duration. The number given in parenthesis represents the ratio $\mu\lambda$.

6.2. Query performances

In this subsection, we examine the performance of the proposed indexing scheme against the Elmasri's time index under a variety of queries.

In the case of the time index, queries relating to both arrival and/or departure time can be supported with reasonable efficiency. For instance, retrieving all persons who arrived between the time interval $[a, b]$ can be accomplished by (i) traversing the B^+-tree to locate the leaf node containing time point a, and (ii) following the sequential links between leaf nodes and retrieving all incremental entries in the buckets, right up to time point b. The efficiency achieved in this instance, however, could be matched by using a V-order B^+-tree. We substantiate this claim by the following analysis.

Let Q_e be the number of page accesses needed to answer the query "Retrieve all persons who arrived between time interval $[a, b]$" using Elmasri's time index. For the purpose of the analysis, we assume that a is a randomly selected time point from interval $[0, now]$, and the length of the interval (i.e. $b - a$) is an exponential random variable with mean γ.

To obtain an estimate of the number of distinct time points in BP which correspond to the given interval, we do the following. Let L_n be the nth order interarrival time for the underlying arrival process. (L_n is an Erlang random variable with mean n/λ.) We are interested in what n might be, assuming that the nth arrival (from time a) coincides with γ (the mean interval value). Hence, we have

$$\frac{n}{\lambda} = \gamma \Rightarrow n = \lambda\gamma$$

The total number of distinct time points (which must include arrivals and departures) which are in BP and which fall into the interval $[a, b]$ is therefore

$$2\lambda\gamma \times \text{Prob}(X > 0) = 2\lambda\gamma\, e^{-\lambda}$$

A conservative estimate of the number of the time index leaf nodes which need to be accessed is therefore $\lceil (2\lambda\gamma\, e^{-\lambda})/(2m_e \ln 2) \rceil$. For each of these leaf nodes, the associated bucket would need to be searched as well. Ignoring any other additional disk accesses (say, fetching the non-leaf nodes of the time index B^+-tree), we obtain

$$Q_e = \left\lceil \frac{\lambda\gamma\, e^{-\lambda}}{m_e \ln 2} \right\rceil \times \left(1 + \frac{\mu\lambda + 2m_e\lambda \ln 2}{2m_e}\right)$$

Again, this value could blow up for large values of $\mu\lambda$.

Consider now the index support provided by a V-order B^+-tree. We denote by Q_0, the number of disk accesses needed to retrieve all temporal data points corresponding to the same query: "Retrieve all persons who arrived during the interval $[a, b]$". As we have suggested earlier, this temporal query can be mapped to a spatial search on the region as indicated in Fig. 11. Assuming that *now* is currently some value n_0, this spatial search can be mapped to at most two interval searches: $(a, 0) \rightarrow (b, n_0 - b - 1)$ and $(a, n_0 - a) \rightarrow (b, n_0 - b)$[1]. In the worst

[1] The reason behind this should be clear by studying the V-ordering as shown in Fig. 8.

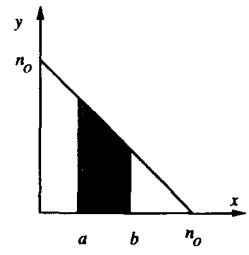


Fig. 11. Search region corresponding to tuples having an arrival time in the interval $[a, b]$.

case, the expected number of points in each of the intervals is bounded by $\lambda\gamma$ (since there are γ time units on the average, and the expected number of arrivals during any one is merely λ). Clearly,

$$Q_0 < 2 \times \left\lceil \frac{\lambda\gamma}{2m_0 \ln 2} \right\rceil$$

Comparing the above page accesses with the time index disk accesses, the V-order B^+-tree clearly provides better and more consistent performances.

Integral to the design of the proposed indexing scheme is the desire to provide a uniform framework for temporal queries of all kinds. In this respect, the proposed scheme distinguishes itself in being able to support a host of other operations which have not received any help from existing indexing schemes such as the time index. For instance, a query such as "Find all persons who stayed in the country for more than 30 days" means scanning the entire database if only existing temporal indexes were used. Using the method which we have proposed, this query corresponds to at most two interval searchers of a H-order B^+-tree.

Note that the main aim of this paper is to propose a scheme for indexing temporal data which could be translated to many different implementations. For this reason, we have chosen to compare different indexes with the time index under different query requirements. This is reasonable as the choice of which index to adopt in a real setting is dependent on the needs of the application. We have demonstrated that for each class of temporal queries, there exists an indexing scheme which outperforms the time index. This clearly provides much more flexibility to system designers to tune the performance of the query system to respond to queries which are most critical or frequently asked. The existence of multiple indexes (using different orderings) moreover provides opportunities for optimization and is analogous to the classical problem of access path optimization.

A plausible alternative to organizing the temporal data points is to adopt some spatial indexing method in place of the classical indexing schemes. This approach would advocate the partitioning of data space into subspaces and organizing these in a hierarchy. This notion of cell partitioning is similar to that of the grid file [9]. However, due to the types of search spaces unique to temporal queries (see Fig. 4) rhombus shaped cells may be more efficient than the conventional rectangular grid cells. Another major difference is that the mapped data space is not bounded but instead grows diagonally. Extensions of the grid file aimed at exploiting these characteristics are currently being studied.

7. Support for complex queries in bitemporal databases

As mentioned earlier, the full benefits of temporal databases can only be realized if data retrieval based on the different notions of time can be supported efficiently. In this section, we demonstrate how the proposed approach can be used to support complex queries in a bitemporal database (a temporal database that supports both valid and transaction time). In other words, we are interested in indexing techniques that can efficiently handle queries that involve both the transaction and valid time. For example, a query of the nature "Find all people who are known to arrive on day 4 as of day 10" (assuming that day 10 has already passed). To our knowledge, none of the existing work address how such queries may be supported.

We use Fig. 12 to illustrate how our proposed indexing scheme can offer huge savings by indexing the points constructed from the four-dimensional space. Here, each tuple μ in the database has associated with it two time intervals: a transaction time interval $[x_s, x_e]$, denoted by $\mathcal{X}(\mu)$, and a valid time interval $[v_s, v_e]$, denoted by $\mathcal{V}(\mu)$.

The following remarks are useful in understanding the content of this relation:

(1) Tuple t1 has been added to the database retroactively, since it is discovered that p1 was in the United States from day 0 to 3 only when he re-enters on day 4.
(2) Person p2 enters the United States on day 0 and leaves on day 5; arrival and departure events result in the tuples t3 and t4, respectively.
(3) Person p3 was mistaken to have left on day 4; the mistake was discovered on day 7 when he actually leaves the country resulting in tuple t7.

Example 5. Consider this query on the `arrival` relation: "Find all persons who are known to be present in the United States on day 2, as of day 3". This query can be transformed to a spatial search on a four-dimensional space by once again, defining a function $\mathcal{M}$ on tuples in

tuple	pid	entry_pt	$\mathcal{V}(\mu)$	$\mathcal{X}(\mu)$
t1	p1	NY	[0,3]	[4,*now*]
t2	p1	LA	[4,*now*]	[4,*now*]
t3	p2	SFO	[0,*now*]	[0,5]
t4	p2	SFO	[0,5]	[5,*now*]
t5	p3	LA	[0,*now*]	[0,4]
t6	p3	LA	[0,4]	[4,7]
t7	p3	LA	[0,7]	[7,*now*]
t8	p3	SFO	[8,9]	[8,*now*]
t9	p4	NY	[2,*now*]	[2,3]
t10	p4	NY	[2,3]	[3,*now*]
t11	p4	LA	[8,*now*]	[8,*now*]
t12	p5	LA	[10,*now*]	[10,*now*]
t13	p6	NY	[12,*now*]	[12,*now*]
t14	p7	NY	[11,*now*]	[11,*now*]

Fig. 12. The `arrival` relation that supports both transaction and valid time.

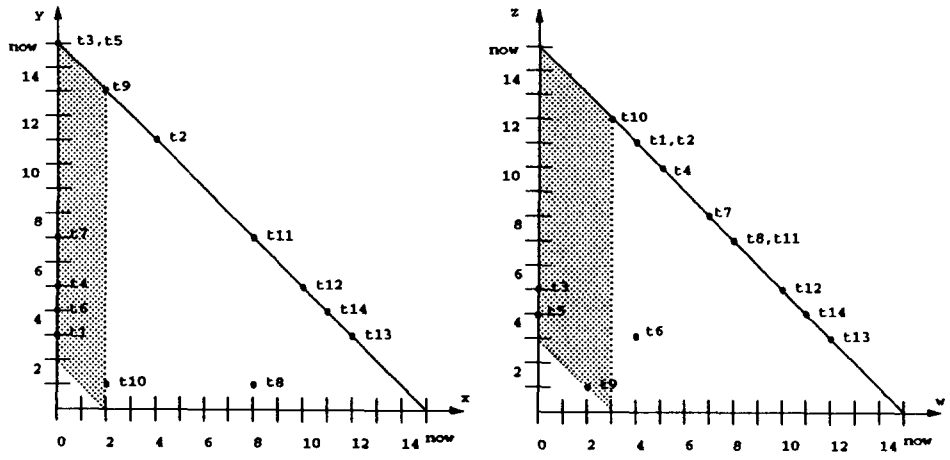


Fig. 13. Four-dimensional search space for the temporal query "Find all persons who are known to be present in the United States on day 2, as of day 3".

arrival. Given any tuple μ, $\mathcal{M}(\mu) = (x, y, w, z)$ where $(x, y) = \mathcal{T}(\mathcal{V}(\mu))$ and $(w, z) = \mathcal{T}(\mathcal{X}(\mu))$. The earlier query can now be answered by identifying those points which are in the four-dimensional search space as shown in Fig. 13. The answer is clearly tuples t3, t5, t9 and t10.

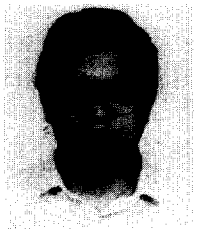
8. Conclusion

In this paper, we addressed the issues of temporal indexing by using existing B^+-trees. The proposed paradigm provides an elegant framework within which temporal selections based on transaction time, valid time, and/or non-temporal attributes are shown to be equivalent to spatial searches in a multi-dimensional space. Index support for these operations can be obtained by merely defining a linear order on temporal data points in this multi-dimensional space, and making use of conventional indexing structures such as the B^+-tree. This enables existing DBMSs to be used for the implementation of temporal databases with little or no modifications. Performance analysis shows that this approach leads to indexes which are more efficient than the time index [4] in both storage utilization as well as query efficiency.

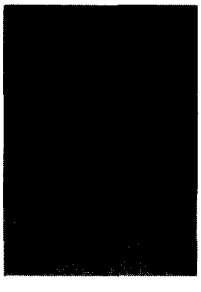
The work reported here can be extended in a variety of ways. Amongst those already mentioned include: (i) identifying extensions to spatial indexes (such as the grid file) for temporal indexing, and (ii) query optimization strategies in situations where different indexes (e.g. index structures created using different orderings) exist. Other work which are currently in progress include: (iii) examining how other temporal operators (such as temporal join) could benefit from this indexing scheme, and (iv) extending an existing DBMS with temporal facilities by augmenting it with a frontend that will provide for mapping between a temporal model and a (non-temporal) backend DBMS.

References

[1] U. Dayal and G. Wuu, A uniform approach to processing temporal queries, in: *Proc. 18th VLDB* (1992).
[2] C.J. ??????, ed., A consensus glossary of temporal database concepts, *SIGMOD RECORD* 23(1) (1994) 52–63.
[3] R. Elmasri and G. Wuu, A temporal model and query language for temporal databases, in: *Proc. 6th Int. Conf. on Data Engineering* (1990) 76–83.
[4] R. Elmasri, G. Wuu and Y.-J. Kim, The time index: an access structure for temporal data, in: *Proc. 16th VLDB* (1990) 1–12.
[5] S. Gadia, A homogeneous relational model and query language for ER databases, *ACM Trans. Database Syst.* 13(4) (1988) 418–448.
[6] S. Gadia and C.-S. Yeung, A generalized model for a relational temporal database, in: *Proc. ACM SIGMOD* (1988) 251–259.
[7] A. Guttman, R-trees: A new dynamic index structure for spatial searching, in: *Proc. ACM SIGMOD* (1984) 45–57.
[8] A. Henrich, H.-W. Six and P. Widmayer, The lsd tree: Spatial access to multidimensional point and non-point objects, in: *Proc. 15th VLDB* (1989) 45–53.
[9] K. Hinrichs, Implementation of the grid file: Design concepts and experience, *BIT* 25 (1985) 569–592.
[10] C. Kolovson and M. Stonebraker, Indexing techniques for historical databases, in: *Proc. 5th Int. Conf. on Data Engineering* (1989) 127–137.
[11] H.-P. Kriegel and B. Seeger, Multidimensional order preserving linear hashing with partial expansions, in: *Proc. 4th Int. Conf. on Data Engineering* (1988) 369–376.
[12] D. Lomet and B. Salzberg, Access methods for multiversion data, in: *Proc. ACM SIGMOD* (1989) 315–324.
[13] V. Lum, P. Dadam, R. Erbe, J. Guenauer, P. Pistor, G. Walch, H. Werner and J. Woodfill, Designing dbms support for the temporal dimension, in: *Proc. ACM SIGMOD* (1984) 115–130.
[14] B.C. Ooi, *Efficient Query Processing in Geographic Information Systems* (Springer-Verlag, 1990).
[15] D. Rotem and A. Segev, Physical organization of temporal data, in: *Proc. Int. Conf. on Data Engineering* (1987) 454–466.
[16] H. Samet, *The Design and Analysis of Spatial Data Structures* (Addison-Wesley, 1989).
[17] A. Segev and H. Gunadhi, Event-join optimization in temporal relational databases, in: *Proc. 15th Int. Conf. on VLDB* (1989) 205–215.
[18] A. Segev and A. Shoshani, Logical modeling of temporal data, in: *Proc. ACM SIGMOD* (1987) 454–466.
[19] N.R.T. Sellis and C. Faloutsos, The R^+-tree: A dynamic index for multi-dimensional objects, in: *Proc. 13th VLDB* (1987) 507–518.
[20] A. Tansel, Modeling temporal data, *Information and Software Technology*, 32(8) (1990) 514–520.
[21] B. Tauzovick, Toward temporal extensions to the entity-relationship model, in: *Proc. 10th Int. Conf. on Entity-Relationship Approach* (1991) 163–179.
[22] G. Wuu and U. Dayal, A uniform model for temporal objected-oriented databases, in: *Proc. 8th Int. Conf. on Data Engineering* (1992) 584–593.
[23] A. Yao, Random 2-3 trees, *Acta Informatica* 2(9) (1978) 159–170.

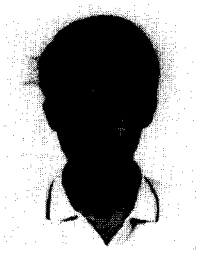


Cheng Hian Goh received his BSc (First Class Honours), and MSc from the National University of Singapore. He is currently a doctoral candidate (Information Technologies) at the Sloan School of Management, MIT. His research interests include heterogeneous database integration, relational database theory, database design, and query optimization.



Hongjun Lu, currently a senior lecturer in the National University of Singapore, received the B.S. degree in electronic engineering from the Tsinghua University, Beijing, China in 1968, and the M.S. and Ph.D. degrees, both in computer science from the University of Wisconsin, Madison. His research interests include data/knowledge base query processing and optimization, parallel and distributed database systems, and knowledge discovery and data mining.

Beng Chin Ooi received B.Sc. (First Class Honours) and Ph.D. in computer science from Monash University, Australia, in 1985 and 1989, respectively. He is currently a senior lecturer at the Department of Information Systems and Computer Science, National University of Singapore. Before joining the Department, he was with the Institute of Systems Science from 1989 to 1991. His research interests include database performance issues, database UI, multi-media databases and GIS. He is the author of a monograph "Efficient Query Processing in Geographic Information Systems" (Springer-Verlag, 1990), a co-editor of three books. He is an editorial board member of the International Journal of Geographical Information Systems (Taylor & Francis), the Journal on Universal Computer Sciences (Springer-Verlag), and the International Journal of Information Technology (World-Scientific). He is a member of Singapore Computer Society, Association for Computing Machinery (ACM) and IEEE Computer Society.



Kian-Lee Tan received his B.S., M.S. and Ph.D. in computer science, from the National University of Singapore, in 1988, 1991 and 1994, respectively. His major research interests include query processing and optimization in multiprocessor and distributed systems, and database performance. He is also a co-author of the tutorial entitled *Query Processing in Parallel Relational Database Systems*. Kian-Lee was a Visiting Scientist at IBM's Almaden Research Center, California (from Jan. 92 to July 92), and CSIRO's Canberra Laboratory, Australia (from June 94 to June 95).