Temporal Relationships in Databases

Surajit Chaudhuri
Stanford University

Abstract

We argue that representation of temporal relationships (e.g., before, after) is necessary in databases. We propose a *graph model* for an important class of temporal relationships. This model is shown to be a powerful tool in identifying generic temporal queries, and in describing the process of deduction of temporal relationships. The model provides a framework to estimate the cost of query evaluation and to identify domain characteristics for query optimization. We provide an outline of temporal query processing to illustrate how domain properties may be utilized. We conclude by presenting an interesting computational model for the temporal domain that trades completeness of the deduction for computational efficiency. Some open problems are mentioned.

1 Introduction

Representation of temporal information and reasoning is necessary in applications such as scheduling, project management, process or device modeling, planning and version management in CAD. The data used by these applications are often voluminous, persistent and need to be shared. Therefore, it is attractive to harness database technology for storage and retrieval of temporal information.

In order to support temporal information management we need to identify the DBMS functionality which would make the task of representation and computation over temporal data simpler. There has been significant work in the area of temporal data modeling [Sno 85], [Gad 85], [Nav 86]. Current models of temporal databases assume that the temporal descriptions associated with an object (or tuple) are specified *absolutely* with respect to a *time-line*[1] (e.g., *25Jan1988* on the timeline of the Gregorian calendar). However, *relative* temporal specifications, based on temporal relationships, are important for many applications. For example, in a planning problem in the blocks world, the support blocks can only be moved *after* the blocks on top of them. Providing for temporal relationships makes the computation over the temporal domain costly. For example, determining the temporal relationship between two events may now require computation of the transitive closure as well as arithmetic calculations.

Although temporal relationships have been studied in program verification, logic programming, and artificial intelligence, there has been little work in examining the problem of supporting the temporal relationships in databases. Enriching the representation of data with temporal relationships raises issues such as what kind of queries may now be expressed and what query processing techniques are appropriate. We briefly discuss these issues in the paper. In order to answer these questions, we propose a simple graph model. The model is shown to be attractive for defining temporal queries, analyzing the complexity of query processing and in identifying the database characteristics that could be used in heuristics for query processing. This paper presents an overview of the graph model and its capabilities. We conclude with a discussion on how the computational complexity of temporal query processing may be reduced significantly by a technique of *weakening deduction*, based on *reference intervals* [All 83].

1.1 Examples of Applications

We first present some examples to motivate the need to represent both relative and absolute temporal information:

[1] A time-line is defined by a unit of time and a scale

Scheduling: The problem of scheduling occurs frequently in computer aided manufacturing and project management [Sat 85]. The result of scheduling is essentially a partial order on individual operations. Causal relationships also induce a temporal relationship (*"if reaction1 causes reaction2, then interval of reaction1 must* precede *that of reaction2"*). Additionally, there are constraints that involve *absolute* time e.g., *"step1 of a process must be initiated no later than 30 seconds after completion of step2"*.

Computer Aided Design: Versions of VLSI designs [Kat 85] are often tagged with temporal information such as *release time* of the design, which indicates the time interval over which the design was the most recent version. A chip uses many components and its design may be best described by a *part-of* hierarchy. For this example, we assume that versions are created only because of changes in the leaf nodes in the hierarchy. The *part-of* relation induces a *containment* relationship between the release times of a component and its parent component in the *part-of* hierarchy. Therefore, to answer the query *"What is the release time of the most recent version of the chip"* we need the ability to represent the containment relationship.

Other Applications: Any model of action requires support for temporal relationships [All 84]. Consequently, examples of temporal relationships could be found in the domain of planning and device modeling [Wil 87]. Story understanding was one of the domains where temporal relationships were first studied [Kah 77]. Since the problem of discovering causal relationships is closely related to temporal ordering, medical information systems also need a similar representation of time [Dow 86].

1.2 Related Work in Databases

Modeling time in databases involves several issues. We mention some of the concepts that are relevant to our work. A comprehensive summary of various research directions in this field may be found in [Sno 86].

Conceptual modeling of *time-line* is an important area of research. There is a long standing debate on whether the right representation of temporal attribute is a point or an interval (or variant thereof). These issues have been dealt in detail elsewhere: [And 82], [All 83], [Lad 86], [Gad 85], [Cli 85] [Nav 86], [Sno 85]. For the purposes of this paper, we do not need to take a stand on this issue. There has also been significant work in the related area is the axiomatization of the temporal structures [All 84], [McD 82].

We are concerned with the computational issues that are involved in supporting temporal domain. Our data model is relational and therefore retrieval is set-oriented. This model of computation is different from the theorem-proving methods that have so far been used in connection with temporal reasoning. Since computation with temporal relationships involves transitive closure, the techniques of recursive query processing may be applied. However, we take a very domain specific view of computation in this paper. The techniques to utilize database characteristics to improve query processing are investigated. We will therefore not discuss the past work in the area of recursive query processing or theorem proving.

The problem of how temporal information may be structured in relational databases is a relevant research issue. In order to appreciate the normal forms that have been developed in this connection, it is important to recognize the two different temporal aspects associated with an attribute.

Temporal Aspects of Attributes: There are two temporal aspects of each attribute [Cli 85]:

1. *Domain:* The domain of the attribute could be time, e.g., date of birth, date of last marriage. Such attributes are called *Time Attributes* (TA). Attributes like name of an employee, rank or salary of an employee are not TAs.

2. *Variability with Time:* An orthogonal property is whether the value of an attribute is a function of time. In case the attribute does not change with time, it is called a *constant attribute* (CA), e.g., name of an employee, date of birth. Otherwise, it is a *time varying attribute* (TVA). The examples of TVAs are date of most recent marriage, rank or salary of an employee.

Observe that the two properties above are orthogonal. For example, while both date of birth and date of most recent marriage are TAs, the former is a CA and the latter a TVA. The normal forms suggest methodologies to group attributes in relations. Examples of such modeling are nested normal form [Cli 85], time normal form [Nav 86] and the homogeneous model [Gad 85]. These different approaches to modeling led to different extensions to the relational model.

Organization of the paper: The rest of the paper is organized as follows. In the following section, we discuss our representation of temporal data (in relational form) and the properties of the temporal relationships. Section 3 is devoted to the introduction of the graph model. In Section 4, examples of the temporal queries are presented and some aspects of query evaluation are discussed. Section 5 investigates how domain specific knowledge may be used to reduce the computational complexity. A hypothesis about temporal applications

is stated, and the method of weakening deduction is discussed. We conclude with some open problems.

2 Representation of Temporal Information

The Problem: We should support relative as well as absolute temporal information. The value of the temporal attribute of a tuple is either known absolutely (numerically) or may be constrained by its temporal relationships (e.g., before, after) to other tuples.
Assumptions: The following simplifying assumptions will be made:

- We restrict ourselves to a subset of binary temporal relationships only. This class will be defined in Section 2.2. Our representation needs to be extended to capture temporal relationships that involve an increment of absolute value, e.g., we can't capture the assertion *"The explosion occurred 4 minutes after the car had left the street"*.
- We are concerned with the representation of relationships between values that have time as a domain. Therefore, one can consider TAs as the focus of our discussion. The domain of time occurs implicitly in TVAs. However, TVAs involve the additional concept of variability and we do not address that aspect here.
- We do not consider the normalization issues here. This simplification is also motivated by our desire to keep the model simple. Therefore, we may view as if no data dependencies (e.g., functional dependencies) are present. We also assume consistency of the database.

A complete model of temporal databases must weaken each of these assumptions. However, these constraints allow us to isolate the issues specific to supporting temporal relationships in a simple way.

Database Schema: We now consider a relational representation for our temporal data. A value for the temporal attribute in a tuple is specified either absolutely or is related to values in one or more tuples symbolically. If there is only one temporal attribute, then we can refer to value of another tuple by its identity (or key). Such a representation fits the description of an *event database*, where events are recorded and every event may have multiple temporal specifications. Accordingly, we refer to our model database as the *"event database"* that includes the following relation that stores the temporal dependencies among events:

$$Temp(e, \mathcal{R}, e')$$

1. e is an event-identity or an absolute temporal value.
2. $\mathcal{R}$ represents the temporal relationship between e and e'; i.e., e is related to e' by $\mathcal{R}$.
3. e' is either an event-identity or an absolute temporal value.

The admissibility of an absolute temporal value depends on the time-line and whether the point or the interval is the primitive concept. For example, when the time-line is the Gregorian calendar, an example of a tuple in *Temp* will be:

$$\langle World\_WarII, <, 1950 \rangle$$

A special case of this representation scheme is when the temporal relationship is restricted to *equality* and the reference is to *absolute* temporal values only. In such a case, the value of the temporal attribute of each event is known *absolutely*. Most of the previous temporal database work assume such a restriction about TAs.

For ease in querying, it may be desirable to represent the information explicitly whether e or e' represents an absolute or relative value. We can add attributes d and d' to *Temp* which encodes respectively whether e and e' is an event-identity or not. Thus, the value of d (or d') is either *absolute* or *event*. However, for the rest of our discussion, we will view the tuples in *Temp* only as triplet. We will implicitly refer to $Temp(e, \mathcal{R}, e')$ by $e\mathcal{R}e'$.

Temporal Relationships as values: In the representation scheme above, we have treated temporal relationship as an attribute instead of a relation. As a consequence, we can talk about properties of the temporal relationships such as subsumption (e.g., $<$ implies $\leq$). Also, we can refer to relationships using quantification (e.g., $\exists$). From the point of view of query processing, computation of a query such as *"What are the temporal relationships between event* a *and event* b*?"* involves reference to a single database relation as against searching multiple relations. This technique of represnting relationships as objects is known as *reification* [Gen 87].

2.1 Properties of Temporal Relationships

We need to specify the class of temporal relationships we support. Therefore, let us consider now the logical properties of temporal relationships. We would like to include properties that capture common temporal relationships. An example of such a property is transitivity.

e.g., *before* is transitive. We restrict the properties of temporal relationships to the set mentioned below as these properties are well understood and help define many temporal relationships. This definition has been motivated by the taxonomy of relationships in [All 83]:

- *Intrinsic Properties:* By intrinsic properties of a relationship $\mathcal{R}$, we mean properties of $\mathcal{R}$ that may be stated without reference to any other relationship:
  - Transitivity (e.g., before, after)
  - Symmetry (e.g., overlapping)
  - Antisymmetry (e.g., before, contained-in)
  - Reflexivity (e.g., overlapping)
- *Extrinsic Properties:* By extrinsic property of a relationship $\mathcal{R}$, we refer to properties that involve reference(s) to at least one other relationship. For many of the temporal relations the extrinsic property is an axiom of the form

$$\forall x, y, z(x\mathcal{R}_1 y \wedge y\mathcal{R}_2 z \rightarrow x\mathcal{R}_3 z) \qquad (1)$$

  where $\mathcal{R}_1, \mathcal{R}_2$ and $\mathcal{R}_3$ are temporal relationships. An example is the following property: *" if x is contained-in y and y precedes z, then x precedes z"*. In this example, x, y, and z designate temporal intervals and the relationship *precedes* between the intervals is defined in the obvious way. We restrict the set of extrinsic properties to (1) only.

Limitations: We have restricted the set of temporal relationships to a set Σ, such that if $\mathcal{R} \in \Sigma$ then $\mathcal{R}$ can be completely characterized by axiom schemata mentioned above. The representation could be based on either points or intervals so long as the above restriction is respected. An example of such a set is $\Sigma = \{<, \leq, >, \geq, =, \neq\}$ over time points. On the other hand, the set $\Sigma_1 = \{before, after, overlaps, incomparable\}$ over intervals is not admissible. This is because if two intervals share a common overlapping interval, they may be related by any one of the relationships as mentioned in Σ_1. One reason for the lack of expressivity is that we have excluded extrinsic properties of the form

$$\forall x, y, z((x\mathcal{R}_1 y) \wedge (y\mathcal{R}_2 z) \rightarrow \bigvee_i (xR_i z))$$

Our axiom schemata enforces that given two events, their temporal relationships are all atomic (no nontrivial disjunction). This makes the form of the derived information same as the base information.

Computational Implications: The logical properties also govern the complexity of query processing. There are some interesting computational implications of these properties:

- *Intrinsic Properties:* The property of a relation being *transitive* immediately suggests that computation of transitive closure will be necessary to answer many interesting queries. The *antisymmetry* property of a relation allows us to infer *equality* constraints among temporal values. It is a simple task to write a relational query to infer these equalities. The property of being *symmetric* implies that when the relation is joined to other relations, it must be augmented such that a tuple $(a\ \mathcal{R}\ b)$ occurs iff $(b\ \mathcal{R}\ a)$ were present in the original relation. This may not be necessary if the query satisfies appropriate symmetry conditions.
- *Extrinsic Properties:* We mentioned earlier in this section that the extrinsic properties will be specified using axiom (1). From database point of view, such an axiom schema implies computation of Generalized Transitive closure. The computation may be cast as a path problem and algorithms for this task has been discussed in [Day 86]. A single application of this axiom results in the following join:

$$\begin{aligned} Result(x, R_3, z) \quad &\leftarrow \quad \sigma_{\mathcal{R}=\mathtt{R}_1} Temp(x, \mathcal{R}, y) \\ &\bowtie \quad \sigma_{\mathcal{R}=\mathtt{R}_2} Temp(y, \mathcal{R}, z) \end{aligned}$$

We have so far seen some implications of the logical properties. But, we need a framework in which we can determine meaningful queries and can also identify the domain characteristics that may be relevant for query processing. In the following section, we present a graph-model to meet these goals.

3 A Graph Model of Temporal Databases

3.1 Database Graph

The primitive concepts that we have considered so far are:

1. Events: Every event is uniquely recognized by an *event-id*.
2. Absolute temporal values: An example is *Jan251988*.
3. Temporal Relationships: Examples are *before* and *after*.

Temporal relationships exist between events and between an event and an absolute value. Any two absolute temporal values on the same time-line are always related.

Corresponding to every event database, we may now construct a graph where nodes are of two sorts (event-id or absolute temporal value) and the directed edges represent temporal relationships. The edges must be labeled to distinguish among various kinds of temporal relationships. Thus, an assertion in the event database is mapped to a labeled edge in this graph. We will refer to this graph as the *database graph*, denoted by G.

3.2 Extended Database Graph

Because of the logical properties of temporal relationships, the database graph G implies additional temporal relationships among nodes in G. The following algorithm augments the graph G such that all such temporal relationships implied by G is present in the resultant graph G':

1. Initialize G' to G.

2. For every three nodes (x, y, z), such that some transitivity axiom applies, add an edge between x and z with an appropriate label. Thus, if we use axiom (1) on a triplet (e_1, e_2, e_3), with ($e_1 \mathcal{R}_1 e_2$), ($e_2 \mathcal{R}_2 e_3$), then we must add an edge from e_1 to e_3 with label $\mathcal{R}_3$.

3. If a relationship $\mathcal{R}$ is symmetric, then for every edge from x to y with a label $\mathcal{R}$, we add an edge from y to x with label $\mathcal{R}$.

4. Repeat steps 1 and 2 till the graph G' does not change any more.

Since we do not delete any edges, G' is independent of the order of application of the transitivity or the symmetry rule. We call G' the *extended database graph*. The algorithm terminates because G is finite and a fixpoint is guaranteed to exist for the horn clause program implied by the logical properties.

Lemma. There exists a relation $\mathcal{R}$ between nodes x and y in G iff there is an edge with label $\mathcal{R}$ between x and y in G'.
□

This algorithm describes the inference of temporal relationships in terms of graph traversal and augmentation. We will discuss the advantage of such a viewpoint in section 3.4.

3.3 Temporal Relationships among Events

Conceptualizing the *event database* in terms of the database graph makes it simple to appreciate the following temporal relationships among events. These relationships should be meaningful across a wide choice of the base set of temporal predicates:

- *Closure of an event:* *Closure* of an event e is the smallest set (call it K) containing e such that

$$y \in K \leftrightarrow \exists \mathcal{R}\ \exists z((y\mathcal{R}z \vee z\mathcal{R}y) \wedge (z \in K))$$

  This set will be denoted by $closure(e)$. The closure of an event is the set of all nodes belonging to the same connected component as the event in the database graph G. This can be computed as the transitive closure of $\pi_{(x,y)} Temp(x, \mathcal{R}, y)$.

- *Restricted Closure of an event:* Our definition of restricted Closure is very similar to Closure, as discussed above. However, instead of quantifying $\mathcal{R}$ existentially, we restrict $\mathcal{R}$ to a *fixed* relationship, e.g, $<$. Thus, restricted closure with respect to R (denoted by $Closure_R(e)$), is the smallest set (call it K) containing e such that:

$$(y \in K) \leftrightarrow \exists z((yRz) \wedge (z \in K))$$

  Hence, $Closure_R(e)$ is the maximal connected subgraph of G' (call it G'_R) containing e such that every edge in it has R as its label. If R is a transitive relationship, then every node in the subgraph G_R is related to e by R. If R does not occur in the right hand side of any extrinsic axioms, then this operation corresponds to the transitive closure on $\pi_{(x,y)}\ \sigma_{(\mathcal{R}=R)} Temp(x, \mathcal{R}, y)$.

- *Weakly related events:* We say that two nodes e and e' are weakly related iff $closure(e) = closure(e')$. For two events to be weakly related they must belong to the same connected component in the database graph. Intuitively, two nodes are weakly related iff the events share some temporal constraint.

- *Independent events:* The extended database graph G' may have more than one connected components. Two events belong to different connected components iff they are not weakly related. We call such events *independent*. A trivial example is when an event $e1$ is specified absolutely, whereas events $e2$ and $e3$ are related by $e2 < e3$. Thus, the number of connected components indicate how many independent sets of events are there in the database.

- *Directly related events:* An important query is "Given two nodes e and e', is there an $\mathcal{R}$ such that $e\mathcal{R}e'$?". If such an $\mathcal{R}$ exists, e and e' are said to be *directly related.* This induces the following property on an event:

  $$Direct(e) = \{e' \mid \exists \mathcal{R}\ (e\ \mathcal{R}\ e' \vee e'\ \mathcal{R}\ e)\}$$

  If two nodes are directly related, then they are also weakly related. The nodes are directly related iff there is an edge between them in the extended database graph G'. In order to evaluate this property, closure needs to be computed.

- *Absolute Bounds:* This is an example of a property that relates temporal attribute of an event to the time-line. Absolute time t is the greatest lower bound on event e iff t is the greatest t' for which $t' \leq e$. Let Z be $closure_{\leq}(e)$. Then, consider the set $Z' \subseteq Z$ such that all members of Z' are absolute nodes. The greatest lower bound is the maximal of the set Z'. If the time-line is bounded between 0 and *now*, then the lower bound corresponds to 0 when Z' is empty. The upper bound may be defined analogously.

- *Closure of an absolute value:* We define closure of an absolute value a, denoted as $closure^{abs}(a)$ as the *closure* of an event, with an added restriction that every member of this set must be an event. In terms of the graph model, $closure^{abs}(a)$ is the set of all event nodes that are in the same connected component as a in graph G_a. G_a augments G by adding an edge between a and every other absolute node a' in G with an appropriate label. Restricted closure of a with respect to R ($closure^{abs}_R$) may be defined similarly.

We have enumerated some of the obvious temporal relationships among events. This list is not exhaustive and illustrates the correspondence between the temporal relationships and the graph properties.

We now establish the correspondence between distance in the graph and the facts on which the derived data depends. Let us consider the case where no extrinsic property is applicable in the database. For example, if the only temporal relationships are: $\{<, =\}$. In that case, the distance between two nodes in G indicate the number of facts on which the deduction of the temporal relationship (weak or direct) is dependent. Of course, thore could be multiple paths between two nodes, each one representing a prrof path.

An issue unaddressed so far is the *consistency* of the database. We must have a way to say that if events x and y are related by a relationship $\mathcal{R}_1$, then x and y can't be related by $\mathcal{R}_2$. In order to express this, we introduce the relation *contradicts* for which the following axiom holds:

$$\forall x, y, \mathcal{R}_1, \mathcal{R}_2 \quad [\ \ contradicts(\mathcal{R}_1, \mathcal{R}_2) \leftrightarrow (x\mathcal{R}_1 y) \rightarrow \neg(x\mathcal{R}_2 y)\ \ ]$$

In terms of the graph model, we can say that if for some temporal relationships $\mathcal{R}_1$ and $\mathcal{R}_2$ $contradicts(\mathcal{R}_1, \mathcal{R}_2)$ is true, then there can't be two edges between two nodes (in the same direction) in G' with labels $\mathcal{R}_1$ and $\mathcal{R}_2$.

3.4 Advantages of the Graph Model:

Viewing our *event database* as a graph is useful in many ways. First, we are able to conceptualize the generic temporal relationships of the last section in terms of simple graph properties. We can also describe the inference of temporal relationships implied by the database graph as the process of edge traversal and edge augmentation of the database graph. Of course, one could directly use the graph algorithms if the temporal relationships are maintained in main memory. Further, an edge traversal or augmentation corresponds to a join operation. This connection between the operations on the graph and relational operators indicates that the graph model can be used in query planning. A complete discussion of the cost model is beyond the scope of the current paper. However, we now mention some of the factors that affect the cost of computation:

1. *Set of logical Properties:* These specify the interdependence of various temporal relationships. Therefore, these partially determine the cost of graph transformation. For example, independence of temporal relationships may be utilized in query planning.

2. *Structure of the database:* Properties of the graph database may be used to characterize the complexity of query evaluation and to decide the applicability of heuristic techniques. In the following section, we discuss how the relative frequency of occurrence of relative versus absolute temporal information may influence query processing. The knowledge of other graph properties may be useful in identifying the termination conditions of algorithms or in strengthening data selection.

3. *Query:* Given a database graph G, and a query, we need to address the following questions:

   (a) What part of extended database G' is the answer to the query?

(b) How to compute the above subset of G' ? In terms of the graph model, we can raise questions such as which search strategy should be followed (i.e., in what order are the axioms to be applied)?

The graph model provides a conceptual framework to analyze the computational aspect of temporal query processing. The following section illustrates the use of the graph model in identifying domain specific heuristics in query evaluation.

4 Temporal Query: Specification and Processing

In this section, we examine how temporal queries may be processed. We provide some examples and describe the computation informally in terms of the graph model. Next, we describe our query language and finally provide a brief overview of query processing.

4.1 Examples of Temporal Queries

- *Relating events and absolute time values:*

  1. Find lower time bounds on the event e: An absolute value t is a lower time bound on an event e iff $t \leq e$. To obtain the greatest lower time bound, we need to compute $Maximal(closure_{\leq}(e))$, restricted to absolute nodes. However, if we want *any* lower time bound, then we need to traverse paths of increasing length from e, only until we reach *an* absolute node in the database graph (instead of all). Such a query is useful in Medical Information Systems, where time of a biological event may have to be inferred from causal relationships.
  2. Find events that took place in between an *absolute* temporal interval (a, b). This is a conjunctive query and can be decomposed into two primitive queries.

     (a) Find set of events that succeed a:

     $$S_1 = closure^{abs}_{\geq}(a)$$

     (b) Find set of events that precede b:

     $$S_2 = closure^{abs}_{\leq}(b)$$

     The answer to the query is $S_1 \cap S_2$. The process analyzer programs that look at the collected process data in a scheduling environment requires this kind of queries. The graph traversal technique outlined in the previous query can be used. However, the computation of S_1 and S_2 should be shared.

- *Relationship between events:*

  1. Temporal relationship between two events e_1, e_2: This query is equivalent to the following subqueries:

     (a) $e_2 \in direct(e_1)$

     (b) $e_2 \in closure(e_1)$

     The second query needs to be executed only if the first query returns no answer. However, the query processing strategy should optimize the queries together. This example has been worked out in section 4.3.
  2. Common ancestors of two events (e_1, e_2): This query is equivalent to computing $closure_{<}(e_1) \cap closure_{<}(e_2)$. However, if only one common ancestor is required, then the computation may be terminated as soon as the first answer is available. As in other cases, the query processing may be easily optimized so that the closures need not be computed independently.

4.2 Query Language

Our goal here is to suggest primitives for expressing queries involving temporal relationships. From section 2, we recall that an atomic formula is of the form $Temp(x, \mathcal{R}, z)$. We have three sorts in our language *event-id, absolute value* and *temporal relationships*. The first and the third argument in *Temp* are restricted to be of the sort event-id or absolute value. The second argument should be of the sort temporal relationship. For any *event database* with a database graph G, a tuple is in *Temp* iff the tuple corresponds to an edge in the corresponding extended database graph G' (derived from extension of G) or the tuple corresponds to a true arithmetic statement over the time-line (e.g., *25Jan1988 before 26Jan1988*).

The specific temporal relationships are user defined. However, the graph model enables us to define *weakly related* (WR) and *independent* (I) as additional temporal relationships. Also, we have two unary functions *lowerbound* and *upperbound* and a predicate *absolute* (which checks whether the object is an absolute value) to apply to objects (variables and constants) of the sort event-id or absolute value.

We denote the atomic queries over *event database* by $(m\ \mathcal{R}_{m,n}\ n)$, where m and n are of the sort event-id or absolute value; and $\mathcal{R}$ is a temporal relationship. Also,

each of (m, $\mathcal{R}_{m,n}$, n) could be either a variable or a constant.

4.3 Query Evaluation

We consider how an atomic query may be evaluated. For simplicity, we assume that no extrinsic properties are present. The main steps in processing the query ($m\ \mathcal{R}_{m,n}\ n$) are as follows:

1. If $\mathcal{R}_{m,n}$ is a constant (R), we do the following operation:
$$\sigma_{\mathcal{R}=R}\ Temp(x, \mathcal{R}, y)$$
2. Depending on the bindings in the atomic formula, one if the following cases apply:
   - Both m and n are constants:
     - m and n are event-ids: A bidirectional search, where we traverse one edge at a time from m and n alternately, works best if there are a large number of absolute nodes in the graph. This is because we can relate two absolute values easily. The bidirectional search implies that the closures be computed in step. Of course, the computation will be shared. However, the answer set will be complete only if no additional independent relationships could exist.
     - Only one of m and n is an event-id: We compute the closure (or restricted closure) of the event-id only if there are large number of absolute nodes. Otherwise, we determine $closure^{abs}(a)$, where a is the absolute constant.
   - Only one of m and n is a constant: To utilize the available bindings, we must start from the constant, whether it is an event or an absolute value, and compute the necessary relationships.
   - Both m and n are unbound: If both m and n are variables of the sort absolute node, then the query is potentially unsafe over an infinite time-line.

The description as above highlights the importance of domain knowledge. The structure of the database graph G plays an important role in query processing. In particular, we considered the database characteristic of relative occurrence of absolute nodes and the knowledge that two absolute values can be related easily. It is an open question to identify other domain characteristics. Examples of promising characteristics are maximal size of a connected component, acyclicity property of the graph. Also, optimization across conjuncts is important so as to prune away large parts of the temporal graph.

5 Domain Knowledge to Improve Query Processing

The last section discussed the use of database characteristics for query processing. However, the problem of computing transitive closures for queries remains a computational bottleneck. Therefore, in this part, we investigate whether there is some common characteristic of many temporal applications that can be used to speed up query processing. We first state a hypothesis and then discuss a technique of *weakening deduction* that provides an interesting computational model,

5.1 Hypothesis about Temporal Applications:

Both Kahn [Kah 77] and Allen [All 83] observed that the temporal information is best kept organized around a small set of *key events*, which serve to index (and partition) the temporal database. The hypothesis that such key events may be identified provides us with a basis for experimenting with heuristic approaches to query processing.

5.2 Weakening Deduction:

One approach to reduce computational overhead is to limit the *heuristic adequacy* of the reasoning system. Limiting heuristic adequacy implies that the reasoning system will be incomplete, i.e., even when a conclusion may be drawn from the database, the reasoning system may not be able to deduce the conclusion. The motivation to limit the reasoning power is to have a cheap computational model that answers most of the *likely* queries, but is not infallible. This idea is based on the *Reference Interval* proposed by Allen [All 83].

Reference Objects: We introduce the concept of *reference objects* in our model. The reference objects are chosen from the set of events and absolute temporal values by a domain expert or by a semiautomated procedure. For every event, a set of reference objects is selected with which the event is *associated*. The reference objects should correspond to *important* events or time-points to which many events are related. Each reference object has a cluster of events, which are all the events associated with it. The clusters for two reference

objects may be intersecting. We assume that if two reference objects are related, then a relationship between them exists that refers to other reference objects only.

Computational Model Reference objects help in guiding the search over the database graph for temporal relationship between two objects. To establish the temporal relationship between any two events, we first search for an *explicit* assertion in the database. If no such assertion is found, then we check whether the events share some common reference object. If so, then their temporal relationship is computed. Otherwise, the temporal relationship between two events across clusters *must always* be established through their reference objects. That means that if events e_1 and e_2 are associated with reference object ro_1 and ro_2 respectively, then the computation for establishing the temporal relationship consists of finding the temporal relationship between e_1 and ro_1, e_2 and ro_2, and finally between ro_1 and ro_2. The relationship between ro_1 and ro_2 can be established via other reference objects only. A formal description follows:

Formal Definition using Graph-Model:

1. Set of reference objects is any chosen subset of nodes in the database graph.

2. Definitions:

   (a) *Cluster(e,r):* This is true iff e is *associated* with the reference object r. Thus, $Cluster(e,r) \rightarrow e \in closure(r)$ and if $Cluster(x,r)$ and $Cluster(y,r)$, then the temporal relationship between x and y will be computed.

   (b) *Transitively correlated :* Two events e and e' are *correlated* iff there is some reference object r such that $Cluster(e,r)$ and $Cluster(e',r)$. e is *transitively correlated* to e' iff there exists a sequence a_i,where $a_0 = e$ and for some m, $a_m = e'$ and for all i, a_i is comparable to a_{i+1}.

3. *Retrieval:* The temporal relationship between any two events may be found by first trying for an explicitly asserted relationship, or else, trying for a temporal relationship via the set of reference objects. More precisely, temporal relationship between two events e and e' may be deduced iff there are temporal associations (e,r) and (r',e'), such that the reference intervals r and r' are *transitively correlated* and every member in the corresponding sequence is a reference interval.

Computational Advantage: We will now give an example to illustrate the computational advantage of this approach. Let us consider an n-node database graph which is a tree. We assume that the database graph is of the form of k disjoint sets of subtrees, i-th set containing n_i nodes. Assume that the reference objects form the root of each subtree. The edges between reference objects form a tree. Thus, there is at most one path between two reference objects. The path represents the relation between the reference objects of the two clusters. Therefore,

$$\sum_{i=1}^{k} n_i = n$$

Finding the temporal relation between two nodes in clusters i and j involve the following:

1. Edge traversals within clusters to relate to the *reference objects.* This amounts to $(n_i - 1) + (n_j - 1)$ traversals.

2. Edge traversals across clusters. This takes $k - 1$ traversals.

Thus, in the worst case, the number of edge traversals in the reference object method is $n_i + n_j + k - 3$. Otherwise, the number of edge traversals is $n - 1$. By a judicious selection of k, one could thus get significant saving in computational cost.

Augmentation of Query Language: In order to support this kind of computational model in our query language, we need three additional primitives:

- $Reference(e)$: This asserts that e is a reference object.

- $Cluster(e,r)$: This asserts that e is associated to the cluster of r.

- $Reference\_related(e1, \mathcal{R}, e2)$: This holds iff the weakened retrieval process yields a temporal relation $\mathcal{R}$ between $e1$ and $e2$. Clearly, this predicate preserves the soundness property of the temporal relationship:

$$Reference\_related(e1, \mathcal{R}, e2) \rightarrow Temp(e1, \mathcal{R}, e2)$$

Limitations of the approach: Since we do not follow all paths between the two nodes in the database graph, such a reasoning system is incomplete. Only a subset of all the relationships between the two events may be deduced:

- Even if there is a temporal relationship between two events, we may not be able to deduce it.

- We may be able to deduce only a *weak* relationship when a stronger relationship exists. A common situation would be even if two events are *directly related*, we may only be able to infer that they are *weakly related*. For example, assume $cluster(ro_1) = \{ro_1, e_1, e_2\}$ and $cluster(ro_2) = \{ro_2, e_3\}$. Let $G = \{e_1 < e_2, e_2 < e_3, e_1 < ro_1, ro_2 > e_3, ro_1 < ro_2\}$. By this method of computation we could only infer that e_1 is weakly related to e_3 although they are directly related because of transitivity of $<$.

However, the domain hypothesis (Section 5.1) assures us that this computational model will be complete most of the time as the interaction between two events belonging to different clusters should mostly be via the corresponding reference intervals.

5.3 Other Approaches

We mention two other approaches to reduce the cost of query processing. Both these are based on the technique of precomputation.

- *Views over Temporal databases:* Selective materialization of views may reduce the amortized cost of query processing over the database [Han 87]. As an example, assume that the information about lower and upper bounds on values of the temporal attribute of the events are kept materialized. Then, while searching for temporal relationship between two events this information may be utilized to prune away the irrelevant part of the database at a low cost by strengthening the selection clause. Of course, materialized views introduce the problem of view maintenance. We need to examine what kind of views are incrementally maintainable [Bla 87] and how best these can be composed [Fin 82]. This work is currently in progress at Stanford University.

- *Main Memory Data Structures:* Appropriate main memory indexing structures may be quite useful. There has been significant work in the past in the area of *path compression* that is relevant. Path compression techniques construct a physical path between any two nodes directly as and when the algorithm discovers the relationship between those. Unfortunately, most of these algorithms, such as in [Ull 73], assume that the input data remain unaltered and inferences are monotonic. This assumption is not valid for temporal database, where data update and inferences are interleaved. However, it is worth investigating to what extent the techniques can be adapted to our problem. Another problem is that main memory structures in path compression algorithms are very tightly associated with the specific queries. This makes it harder to directly use these data structures for general indexing of temporal data.

6 Conclusion

Many applications require representation of temporal relationships. In this paper, we proposed that such temporal relationships be supported in relational databases. The following issues related to such representation have been considered:

- A common set of logical properties of temporal relationships.

- A graph model that serves the following purposes:
  - Representation of the database assertions as a labeled graph.
  - Graph transformations and path traversals describe the deductions implied by the properties of the temporal relationships.
  - A framework to estimate cost of computations and to identify domain characteristics that are useful for query evaluation.

- An outline of query processing illustrates how relative occurrences of *absolute values* influences evaluation heuristics.

- A computational model based on *reference objects* [All 83]. Our proposed approach trades completeness of deduction for significant reduction in computational complexity.

Further work is necessary to characterize our approach completely and to extend the techniques presented in this paper. We mention some of our immediate interests:

- Determination of the class of temporal properties (over points and intervals) that can be captured by the graph model, as it is now defined.

- Development and verification of a cost model for Temporal Query evaluation, based on the graph model.

- Evaluation of the temporal view approach.

- Extension of the model to capture a wider class of temporal relationships, such as periodicity and delay.

To summarize, our approach has been to admit temporal relationships and to examine the computational requirements imposed by such representation. We believe that the above is a pragmatic approach to modeling of temporal databases.

7 Acknowledgments

Prof. Gio Wiederhold is responsible for encouraging me to work on the area of temporal databases. Comments and suggestions from Waqar Hasan, Peter Rathmann, Prof. Wiederhold and the reviewers were valuable. The author appreciates the comments from Arun Swami, Arthur Keller, Marianne Winslett and Matt Morgenstern on an earlier version of the paper. This work was performed as part of the KBMS project, supported by DARPA Contract No. N00039-84-C-02111.

References

[Ahn 86] Ahn, I. "Towards an implementation of Database Management Systems with Temporal Support"; IEEE CS Data Engineering Conference, 1986.

[All 83] Allen, J. F. "Maintaining Knowledge about Temporal Intervals"; CACM, Nov 1983.

[All 84] Allen J. F. "Towards a General Theory of Action and Time"; *Artificial Intelligence 23*, 1984.

[And 82] Anderson, T.L. "Modeling Time at the Conceptual Level"; in *Improving Database Usability and Responsiveness*, P. Scheurmann (Ed.), Academic Press, 1982.

[Bla 87] Blakeley, J. A. "Updating Materialized Database Views"; University of Waterloo Research Report, May 1987.

[Cli 85] Clifford, J. "On an algebra for Historical Relational Database: Two views"; Clifford James and Tansel A.U., Proceedings of ACM SIGMOD, 1985.

[Day 86] Dayal, Umeshwar and Smith John Miles; "PROBE: A Knowledge-Oriented Database Management System"; in *On Knowledge Base Management Systems*, Ed. Brodie, M.L and Mylopoulos, J, Springer-verlag, 1986.

[Dow 86] Downs, S. M. et al "Automated Summarization of On-line Medical Records"; MedInfo 1986.

[Fin 82] Finkelstein, S. "Common Expression Analysis in Database Applications"; Proceedings of ACM SIGMOD 1982.

[Gad 85] Gadia, S.K. and Vaishnav J.H. "A Query Language for a Homogeneous Temporal Database"; in Proceedings of the ACM symposium on Principles of Database Systems, Apr. 1985.

[Gen 87] Genesereth, M. and Nilsson, N. *Logical Foundations of Artificial Intelligence*; Morgan Kaufman Press, 1987.

[Han 87] Hanson, E. N. "A Performance Analysis of View Materialization Strategies"; Proceedings of ACM SIGMOD, 1987.

[Kah 77] Kahn, K. et al "Mechanizing Temporal Knowledge"; *Artificial Intelligence 9*, 1977.

[Kat 85] Katz, R. et al; "Organizing a design database across time"; Workshop on Large Scale Knowledge Base and Reasoning Systems, Feb 1985.

[Lad 86] Ladkin, P. "Time Representation: A Taxonomy of Interval Relations"; Proceedings of AAAI 1986.

[McD 82] McDermott, D.V. "A Temporal Logic for Reasoning about Processes and Plans"; *cognitive Science 6*, 1982.

[Nav 86] Navathe, S. and Ahmed R. "A Temporal Relational Model and a Query Language"; Working paper, 1986.

[Sat 85] Sathi, A., Fox, M. and Greenberg, M. "Representation of Activity Knowledge for Project Management"; *IEEE Transactions on Pattern Analysis and Machine Intelligence*, September 1985.

[Sno 85] Snodgrass, R., Ahn I; "A Taxonomy of Time in Database"; Proceedings of ACM SIGMOD, 1985.

[Sno 86] Snodgrass, R. (Ed.) "Research Concerning Time in Databases: Project Summaries"; ACM SIGMOD RECORD, Vol 15, No. 4, December 1986.

[Ull 73] Ullman, J.D, Hopcroft J.E and Aho A.V. "On Finding Lowest Common Ancestors in Trees"; Proceedings of ACM Symposium on Theory of Computing, 1973.

[Wil 87] Williams, B. "Doing Time: Putting Qualitative Reasoning on Firmer Ground"; Proceedings of IJCAI 1987.