A Technique for Generalizing Temporal Durations in Relational Databases

D.J. Randall, H.J. Hamilton, R.J. Hilderman
University of Regina
Regina, Saskatchewan, Canada, S4S 0A2
{randal,hamilton,hilder}@cs.uregina.ca

Abstract

We address the problem of generalizing temporal data concerning durations extracted from relational databases. Our approach is based on a domain generalization graph that defines a partial order specifying the generalization relations for a duration attribute. This domain generalization graph is reusable (i.e., can be used to generalize any duration attribute), adaptable (i.e., can be extended or restricted as appropriate for particular applications), and transportable (i.e., can be used with any database containing a duration attribute).

Introduction

In current research, [1] we are investigating methods for *temporal generalization*, i.e., generalizing a data set including at least one temporal attribute by replacing the given temporal values with more general temporal values. Temporal generalization is useful for data mining from databases that include temporal attributes. In our research on data mining from relational databases, we have discovered that the *time* domain is a common one for attributes, but that generalization of temporal values requires close attention to the meaning of these values. Although recent advances have been made in the management of temporal data in relational databases (11), less research has focused on the development of techniques for generalizing and presenting temporal data (8).

A temporal attribute may be used to specify the beginning of an event (such as a *birth date* to mark the start of a new life), the end of an event (such as the *check out time* from a hotel), or the duration of an event, such as the *elapsed time* of a race. We refer to any attribute whose domain contains date and time values, such as birth dates and check out times, as a *calendar attribute*, and one containing durations as a *duration attribute*. The values for either calendar and duration attributes can be generalized in different ways and to different levels of granularity (1; 4; 10). For example, given a set of running times for a marathon race (a duration attribute), these values could be generalized in one way to give values to the nearest number of quarter hours for the race, or in a second way to one of three categories (fast, medium, and slow). Choosing a different level of granularity for the first alternative could give values generalized to the nearest number of hours.

In this paper, we propose a method for generalizing a duration attribute using domain generalization graphs; in a companion paper (14), we address the problem of generalizing calendar attributes. We define a domain generalization graph (6) for duration attributes by explicitly identifying the domains appropriate for the relevant levels of temporal granularity and the mappings between the values in these domains. Generalization is performed by transforming values in one domain to another, according to directed arcs in the domain generalization graph. Our goal is to specify a domain generalization graph for the duration attribute that is reusable (i.e., can be used to generalize any duration attribute), adaptable (i.e., can be extended or restricted as appropriate for particular applications), and transportable (i.e., can be used with any database containing a duration attribute). Given the duration domain generalization graph and a duration attribute, our method creates a high-level map of the distinct, nontrivially different possible summaries. Each summary is based on a different way of generalizing or a different level of time granularity. This map can be used as the basis for further automatic data mining or presented to the user, who then can choose summaries to examine in more detail.

The remainder of this paper is organized as follows. In the following section, we discuss related work, giving an overview of domain generalization graphs and some temporal theories. In **Generalizing Duration Values**, we explain how to construct a domain generalization graph for a duration attribute. In **Adaptation of the Duration DGG**, we outline a semi-automated method for determining what portion of the domain generalization graph is not valuable to the user. This method can be run in a completely automated fashion or it can receive additional input from the user at two key stages. In **A Language For Specifying Dura-**

tion DGGs, we present a language for specifying the elements of a domain generalization graph for a duration attribute. In **An Extended Example**, we provide a complete example showing how the domain generalization graph can be configured in a semi-automatic fashion for a particular generalization application to constrain which results from a discovery task are shown to a user.

Related Work

Generalization for Knowledge Discovery

A Concept Hierarchy (CH) associated with an attribute in a database is represented as a tree, where leaf nodes correspond to the actual data values in the database, intermediate nodes correspond to more general representations of the data values, and the root node corresponds to the most general representation of the data values. Knowledge about higher-level concepts can be discovered through a general-to-specific search beginning at the leaf nodes using a process called ***attribute-oriented generalization*** (2; 7).

If several CHs are associated with the same attribute, meaning knowledge about the attribute can be expressed in different ways, common attribute-oriented generalization methods require the user to select one CH, ignoring the relative merits of other possible generalizations which could produce interesting results. To facilitate other possible generalizations, ***domain generalization graphs*** (DGGs) have recently been proposed (6; 9; 13).

A DGG is defined as follows (6). Given a set $S = \{s_1, s_2, \ldots, s_n\}$ (the domain of an attribute), S can be partitioned in many different ways, for example $D_1 = \{\{s_1\}, \{s_2\}, \ldots, \{s_n\}\}$, $D_2 = \{\{s_1\}, \{s_2, \ldots, s_n\}\}$, etc. Let D be the set of partitions of set S, and $\preceq$ be a binary relation (called a ***generalization relation***) defined on D, such that $D_i \preceq D_j$ if for every $d_i \in D_i$ there exists $d_j \in D_j$ such that $d_i \subseteq d_j$. The generalization relation $\preceq$ is a partial order relation and $\langle D, \preceq \rangle$ defines a partial order set from which we can construct a ***domain generalization graph*** $\langle D, E \rangle$ as follows. First, the nodes of the graph are elements of D. And second, there is a directed arc from D_i to D_j (denoted by $E(D_i, D_j)$) iff $D_i \neq D_j$, $D_i \preceq D_j$, and there is no $D_k \in D$ such that $D_i \preceq D_k$ and $D_k \preceq D_j$. The partial order set $\langle D, \preceq \rangle$ is transitively closed and is a lattice.

Temporal Representation and Reasoning

Most work related to time in Artificial Intelligence (AI) has concentrated on defining a time domain and facilitating reasoning about events. Time can be represented using timepoints, time intervals, or a combination of the two, and it can be linear, non-linear or branching (8). For application to KDD, where our time domain is based on database timestamps we use timepoints as a basis for defining our DGG and intervals (the timepoints are timestamps in database technology). This means that for our purposes, time is defined as linear, discrete, and bounded (5).

Cukierman and Delgrande (3) construct a calendar structure and calendar expressions which forms the basis for our definition of a calendar DGG (14). They also define four types of decomposition for reasoning about schedulable, repeated activities: aligned and non-aligned; constant and non-constant. ***Aligned decomposition*** means that the union of the submembers completely covers the original member (i.e., days from months). ***Non-aligned decomposition*** means that the union of the submembers does not completely cover the member leaving gaps at the beginning and/or end (i.e., weeks from months). ***Constant decomposition*** means that every time the member is broken into submembers, a constant ratio is maintained (i.e., days from weeks). A ***non-constant decomposition*** has a variable number of submembers (i.e., days from months).

For generalization we define ***compositions***, which are the inverses of Cukierman and Delgrande's decompositions. Four types of compositions can be defined analogously as ***aligned composition***, ***non-aligned composition***, ***constant composition***, and ***non-constant composition***.

Generalizing Duration Values

We now describe our proposed duration DGG for duration attributes, a part of which is shown in Figure 1. The underlying meaning for durations is an amount of time. In the duration DGG, events are grouped together if they have the same duration (or are within some range of durations), regardless of when they happen. In Figure 1, the node labelled *one second (SPECIFIC)* represents the most specific domain considered; i.e., the finest granularity of our duration domain is one second. Higher-level nodes represent generalizations of this domain. The arcs connecting the nodes represent generalization relations.

This is the typical degree of accuracy for timestamps in current commercial database products (12). Durations are commonly stored in two formats. Durations can be shown as *YYYYMMDDhhmmss*, or they can be regarded as a fixed number of seconds.

In many cases, durations are stored in a database by storing the start and end time for an event. In some cases this data is valuable. Months are non constant decompositions. To properly calculate that an event starting on January 23, 1997 and ending February 14, 1997 spans two months, we must know the start date. In the example discussed later, we assume the duration has been computed and stored as a new attribute representing the difference between the two.

We have defined a basic duration DGG as a starting point for generalizing duration attributes. There are two obvious ways to define the the range of values to generalize durations to. A ***constant composition*** can be specified by an interval width, w. The intervals will then be the set: $\{[kw, (k+1)w) : k \in N\}$. The second possibility is an ***nonconstant composition***, specified in detail by listing endpoints which specify disjoint intervals.

We have defined only one nonconstant composition in our duration DGG. While they may facilitate valuable generalizations, they are also data and task specific. We leave it to the user to specify more nonconstant compositions if their data is appropriate.

Adaptation of the Duration DGG

To guide the user to the most interesting information as quickly as possible, we prune the duration DGG. The user can manually prune nodes both before and after the automatic pruning. As well, based on a superficial examination of the data, the duration DGG can be pruned to remove nodes and arcs that are not likely to be of value to the user. The resulting DGG can be shown to the user as a guide to the possible generalizations of interest.

As mentioned in **Generalization for Knowledge Discovery**, any DGG includes at least two nodes. To add a node to an existing DGG, one must identify at least one parent node and at least one child node in the existing DGG and define the edges that connect the parent(s) to the new node and connect the new node to the child(ren). This may require the creation of new tables or functions.

We propose generating the duration DGG based on the data in the following manner. A node in the DGG represents a partition of the set of durations into blocks of time (disjoint subsets). For a duration DGG to present useful information to the user, the number of blocks in a particular partition in a node must be fairly small and a majority of the blocks must be populated. If there are too many blocks the user will be overwhelmed by the volume of data. Alternatively, if there are too few blocks, or if few of them are populated, the valuable information may be lost by over-generalizing. Given a chosen set of equally sized blocks, the block size, the number of blocks, and the minimum duration present in the data are factors which affect our choice of blocks.

The duration DGG is pruned in four steps.

1. Reachability Pruning: Once the user has specified how the data relates to a starting node in the DGG (this node becomes the "SPECIFIC" node), we prune all nodes which can no longer be reached from that node.

2. Preliminary Manual Pruning: The user many prune any of the remaining interior nodes. They are not allowed to prune the node linked with the original data and the "ANY" node.

3. Data Range Pruning: Each node that does not distinguish between any of the duration values in the tuples and all nodes derived from it excepting the "ANY" node are pruned from the graph. For edges that represent a monotonic function, we can inexpensively calculate whether the node will distinguish between duration values in the tuples. Tuples representing the minimum and maximum are generalized along all data range prunable edges in the DGG (those marked

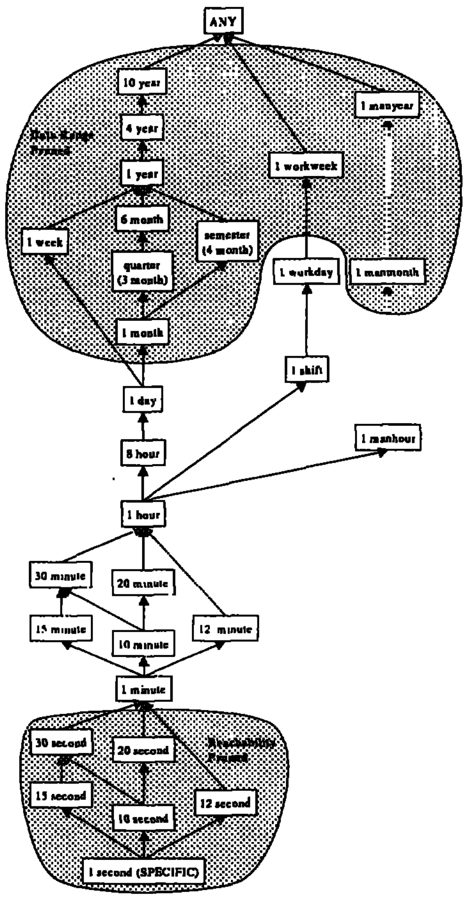

Figure 1: Duration DGG

with a dot in Figure 1). If the minimum and maximum are generalized to a single value, all duration values in the data would be generalized to that value.

4. Pregeneralization Manual Pruning: The user is given the opportunity to perform manual pruning before the tuples are generalized.

A Language For Specifying Duration DGGs

For DGGs to be easily extended, the structures which store the information must be easily editable by the user. To achieve this end we have chosen to use plain text files to store the DGG structure. The grammar is in Figure 2.

To specify a DGG in a text file, we list the edges comprising the DGG. Specifying the edges implies the nodes. For each edge in the DGG, we specify three pieces of information: a unique label for the edge in the DGG, the type of mapping that the edge represents, and the data needed to perform the mapping itself. The edges are uniquely identified by their parent/child pair resulting in a name such as "minutes:hours". The first line describing an edge in a DGG contains this infor-

<DGG> → <Edge> {<Edge>}

<Edge> → <ParentChild> <DGGComponent> {<DGGComponent>}

<ParentChild> → <string> : <string>

<DGGComponent> → <width> | <interval>

<interval> → <number>, <number> {<interval>}

<width> → <number>

<string> → [0-9a-zA-Z] {[0-9a-zA-Z]}

<number> → [0-9] {[0-9]}

Figure 2: DGG Grammar

mation.

The DGGComponent in Figure 2 can be either a width, or a list of intervals.

minutes:hours
60

Table 1: "minutes:hours" Lookup Table

To calculate hours from minutes, the entries in Table 1 are used. Table 1 defines the generalization represented by the nodes shown in Figure 3. The nodes represent the data at some level of generalization while the edges represent the process. When defining a DGG, it is the process we define as these edges, and the data follows as a consequence. That is, we define what new domain values are for the program, but it is up to the program to create instances of them if necessary for given input data.

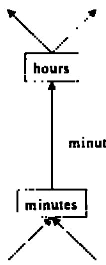

Figure 3: DGG Nodes with Edge

An Extended Example

In this section, we describe an application of knowledge discovery using duration attribute. For conciseness, we emphasize the summary of information with regard to a single attribute rather than combinations of generalized attributes (for which see (6)).

As a source of data, the Unix 'last' program was used. Some sample output is shown in Figure 4. This system utility shows the login times and the duration that a user was logged in for in an "hours:minutes" format. This means that the duration values were available in YYYYMMDDhhmm format, so we do not have second-resolution information to work with. Also the logout time is only given relative to the login time. To calculate the YYYYMMDD information we have to add the duration to the login time given. The main expectation is that we will see a distribution of many short logins and fewer longer logins.

```
username date         in   -  out   duration

user328  Sun Jan 18 00:04 - 00:04  (00:00)
user602  Sun Jan 18 00:48 - 00:48  (00:00)
user645  Sun Jan 18 01:21 - 01:42  (00:20)
...
user723  Sun Jan 18 21:40 - 10:39  (132:58)
...
user328  Sat Jan 24 23:55 - 23:59  (00:03)
```

Figure 4: Sample 'last' output

The user identifies the starting node, "1 minute". To adapt the duration DGG we immediately perform reachability pruning. Because there is no second information, we can immediately prune everything below "1 minute". At this point the user is given an opportunity to manually prune nodes out of the duration DGG.

Next we perform data range pruning. The tuples containing the minimum and maximum duration values are extracted from the data. The minimum is 0 minutes and the maximum is 7,978 minutes (5 days, 12 hours, and 58 minutes), respectively. These are generalized through the duration DGG. We find that the two tuples generalize to a single one at the *month* and *manmonth* nodes (see Figure 1). We prune these nodes and all their children (except the "ANY" node).

Duration (hours)	Count
0 - 1	8502
1 - 2	427
2 - 3	193
3 - 4	57
4 - 5	18
5 - 6	8
...	
51 - 52	1
93 - 94	1
132 - 133	1

Table 2: Login Duration and Count Table

After the user is finished any further manual pruning, we generalize the data. The adapted duration DGG and can be presented to the user as a guide to all relevant generalizations. In Table 2 we present a table showing the data generalized to the "1 hour" level. This is a point where there are few enough tuples that they can be conveniently viewed by the user. Additionally, building on previous work in (14), we have Table 3 showing login times and duration generalized by the calendar DGG (14) to weekday/weekend and by duration to the "1 hour" level. This table also shows weighted % share, which is a scaled percentage so that we can compare the counts for 5 days with counts that are only for 2 days. We can see from the percentages, that the distribution of login durations is independent from the login occurring on a weekday or weekend.

weekday/weekend	hours	count	% share
weekday	0 - 1	6891	92.21
weekday	1 - 2	328	4.39
weekday	2 - 3	142	1.90
weekday	3 - 4	37	0.50
weekday	4 - 5	15	0.20
weekday	5 - 6	8	0.11
weekday	6 - 7	12	0.16
weekday	7 - 8	6	0.08
...			
weekday	93 - 94	1	0.01
weekend	0 - 1	1611	89.50
weekend	1 - 2	99	5.50
weekend	2 - 3	51	2.83
weekend	3 - 4	20	1.11
weekend	4 - 5	3	0.17
weekend	6 - 7	6	0.33
weekend	7 - 8	2	0.11
...			
weekend	132 - 133	1	0.06

Table 3: Login Duration Comparison Between Weekends and Weekdays

Conclusion

We have presented a formal specification for the components of a domain generalization graph associated with a duration attribute. Four different generalization types were discussed and examples given of their implementation. The duration DGG is extensible, allowing the user to easily add new edges and nodes to the DGG when knowledge about a duration attribute can be expressed in different ways. The DGG can be transported to other databases. Examples of the generalizations were given.

Future research involves combining interestingness measures with the summaries presented to the user.

References

[1] C. Bettini, X. S. Wang, and S. Jajodia. A General Framework and Reasoning Models for Time Granularity. *Proceedings of the Third International Workshop on Temporal Representation and Reasoning (TIME-96)*, pages 104-111, Key West, Florida, May 1996.

[2] C.L. Carter and H.J. Hamilton. Efficient attribute-oriented algorithms for knowledge discovery from large databases. *IEEE Transactions on Knowledge and Data Engineering.* To appear.

[3] D. Cukierman and J. Delgrande. A language to express time intervals and repetition. *Proceedings of the Second International Workshop on Temporal Representation and Reasoning (TIME-95)*, pages 41-48, Melbourne, Florida, April 1995.

[4] J. Euzenat. An algebraic approach to granularity in time representation. *Proceedings of the Second International Workshop on Temporal Representation and Reasoning (TIME-95)*, pages 147-154, Melbourne, Florida, April 1995.

[5] J. Gamper. A temporal reasoning and abstraction framework for model-based diagnosis systems. PhD thesis, RWTH Aachen, 1996.

[6] H.J. Hamilton, R.J. Hilderman, and N. Cercone. Attribute-oriented induction using domain generalization graphs. In *Proceedings of the Eighth IEEE International Conference on Tools with Artificial Intelligence (ICTAI'96)*, pages 246-253, Toulouse, France, November 1996.

[7] J. Han, Y. Cai, and N. Cercone. Data-driven discovery of quantitative rules in relational databases. *IEEE Transactions on Knowledge and Data Engineering*, 5(1):29-40, February 1993.

[8] P. J. Hayes. A Catalog of Temporal Theories. Tech report UIUC-BI-AI-96-01, Beckman Institute and Departments of Philosophy and Computer Science, University of Illinois. 1995. Available at http://www.coginst.uwf.edu/~phayes/ TimeCatalog1.ps and TimeCatalog2.ps

[9] R.J. Hilderman, H.J. Hamilton, R.J. Kowalchuk, and N. Cercone. Parallel knowledge discovery using domain generalization graphs. In J. Komorowski and J. Zytkow, editors, *Proceedings of the First European Conference on the Principles of Data Mining and Knowledge Discovery (PKDD'96)*, pages 23-35, Trondheim, Norway, June 1997.

[10] J. R. Hobbs. Granularity. *International Joint Conference on Artificial Intelligence (IJCAI'85)*, pages 432-435, Los Angeles, California, 1985.

[11] N.A. Lorentzos and Y.G. Mitsopoulos. SQL extension for interval data. *IEEE Transactions on Knowledge and Data Engineering*, 9(3):480-499, May/June 1997.

[12] Oracle Corporation. *Programmer's Guide to the Oracle Call Interface*, Release 7.3. 1996.

[13] W. Pang, R.J. Hilderman, H.J. Hamilton, and S.D. Goodwin. Data mining with concept generalization graphs. In *Proceedings of the Ninth Annual Florida AI Research Symposium*, pages 390-394, Key West, Florida, May 1996.

[14] D.J. Randall, H.J. Hamilton, and R.J. Hilderman Generalization for Calendar Attributes Using Domain Generalization Graphs *Fifth International Workshop on Temporal Representation and Reasoning (TIME'98)*, Sanibel Island, FL, May, 1998.