Combining Simultaneous Values and Temporal Data Dependencies

Avigdor Gal* & Dov Dori
Information Systems Engineering Department
Faculty of Industrial Engineering and Management
Technion - Israel Institute of Technology
Haifa, 32000, Israel

Abstract

In temporal databases there are situations where multiple values of the same data item have overlapping validity times. In addition to the common case of multi-valued properties, there are several possible semantics to multiple values with overlapping validity times of the same data item. We refer to such data items as having simultaneous values. This paper presents a polynomial algorithm for efficient handling of simultaneous values in a database with temporal data dependencies—integrity rules that define relationships among values of different data items in a temporal database. The algorithm is demonstrated using a case study from the game theory area. An implementation of the algorithm is integrated in a prototype of a temporal active database.

keywords: temporal databases, simultaneous values, uncertainty, temporal data dependencies, action reasoning

1 Introduction and Motivation

A *temporal database* is a database that supports some aspects of time [5]. One of the basic temporal aspects supported by many temporal databases is the *valid time*, representing the time a data-item is considered to be true in the modeled reality [5].

There are situations where multiple values of the same data-item have overlapping valid times. The multi-valued property is the most common case, where several values are grouped into a single property [4]. For example, a property that contains the languages that a person speaks, can have a set of values grouped into a single property. There are situations, however, where multiple values with overlapping valid times of the same data-item exist in the database, but with different semantics than the multi-valued case. We refer to these data-items as having *simultaneous values*. While in the multi-valued case **all** values whose valid time include t are deemed to be valid in the modeled reality, it is possible that only **part** of the candidate values, i.e. the values that were assigned to a data-item at time t, represent the data-item's value in the real world. For example, a data-item that contains a spouse name is limited by law to be single-valued. When there are several alternatives for the spouse name due to uncertain information, then only one value of the set is the data-item's value. Each value of the set is **possibly** the data-item's value. In temporal databases, change of decisions about the value and valid time of a data-item may cause a situation where two values of the same data-item have overlapping valid times. For example, a value val_1 valid during [Jan 1994, Mar 1994) of a data-item δ is augmented at time point Aug 1993 by a value val_2 valid during [Feb 1994, Apr 1994). δ has more than one value in the interval [Feb 1994, Mar 1994). In some cases, the value that was inserted later corrects an erroneous value that was inserted earlier. In other cases, both values are possibly correct, each with respect to a different time point.

Simultaneous values enable different semantics in mapping the stored values to the modeled reality values. It is particularly useful in applications where a data-item may have multiple values representing the existence of different alternatives. For example, if knowledge arrives from various sources, then no a-priori selection of a single value should be enforced. Instead, for each database retrieval operation, the user can choose the appropriate value, values or any aggregation of those values.

Handling simultaneous values in a temporal database requires the use of optimized update and retrieval mechanisms. The maintenance problem of simultaneous values becomes more arduous in *temporal active databases* [2], where temporal data dependencies are enforced. A *temporal data dependency* is a tool that supports rules for manipulating data-items which may have a variety of temporal characteristics. Temporal data dependencies can be viewed as a type of integrity rules of the temporal active database. Violating them, activates database operations that react to restore the database integrity.

As an example for the use of temporal data dependencies, we can consider *decision support systems* [3]—systems that model decisions about actions that should be performed in a target system. Such systems consist of decision models that are rooted in the operations research or artificial intelligence disciplines, and of a database, that stores the necessary data to support the decision models. Decision support systems can benefit significantly from the temporal active paradigm.

*The work was conducted while the author was in the Technion. He is currently at the Department of Computer Science, University of Toronto, Toronto, Ontario, M5S 3H5 CANADA.

As a concrete motivating case study, we present the following application of a decision support system, based on the Cournot game [7]; [1]. Three instant coffee manufacturers—Bilbo, Frodo and Gandalf, decide each month about the quantity of coffee to be produced in the next month. Each manufacturer bases the decision about its manufactured quantity upon estimation of the quantities manufactured by the other two manufacturers, its own strategy (maximum revenue, a certain market share, etc.), and general knowledge about the market behavior. Each manufacturer has its own deadline for making the production quantity decision. We assume a single market price for the manufactured type of instant coffee, which is determined periodically as a function of the total quantity produced in that period. The relationships between the market price and the total quantity is modeled by the constraint

$$\mathsf{Total-Quantity} \cdot \mathsf{Market-Price} = \mathsf{Market-Constant} \quad (1)$$

Each manufacturer attempts to estimate the best decision to be taken, based on its own competition strategy and the two competitors' decisions and competition strategies. For example, Bilbo may assume that Frodo and Gandalf have an objective of maximum profit. Consequently, each manufacturer would like to produce as much coffee as possible without lowering the price to a level that decreases its total profit. By assuming the competition strategy of both Frodo and Gandalf, Bilbo can estimate their production decisions and determine the optimal production level, based on the following temporal data dependency:

$$\mathsf{Production\text{-}Decision} := \sqrt{\frac{\mathsf{Market\text{-}Constant} \cdot \mathsf{Competitors\text{-}Total\text{-}Estimation}}{\mathsf{Unit\text{-}Cost}}} - \mathsf{Competitors\text{-}Total\text{-}Estimation}$$

This temporal data dependency is a periodical result of maximizing the profit function of a single manufacturer. The derivation of the temporal data dependency is given in Appendix A. Other strategies would yield different temporal data dependencies. Competitors-Total-Estimation is the sum of the production estimations of the other two manufacturers. Since this information is often misleading, each manufacturer should collect as much estimations as possible on each one of the competitors. The temporal dependency graph can be evaluated each time there is a change in one of the data-items Market-Constant, Competitors-Total-Estimation or the manufacturer's Unit-Cost. Alternatively, it can be evaluated once each period, just before the manufacturer has to decide about the quantity to be produced for the following period.

Figure 1 presents the data over time of the data-items Market-Constant, Competitors-Total-Estimation, Unit-Cost, and the resulting Production-Decision of Bilbo. As the figure shows, the temporal validity of each value is bounded. For example, Market-Constant has the value 10000 during the interval [Feb 94, June 94).[1] The resulting values of Production-Decision are

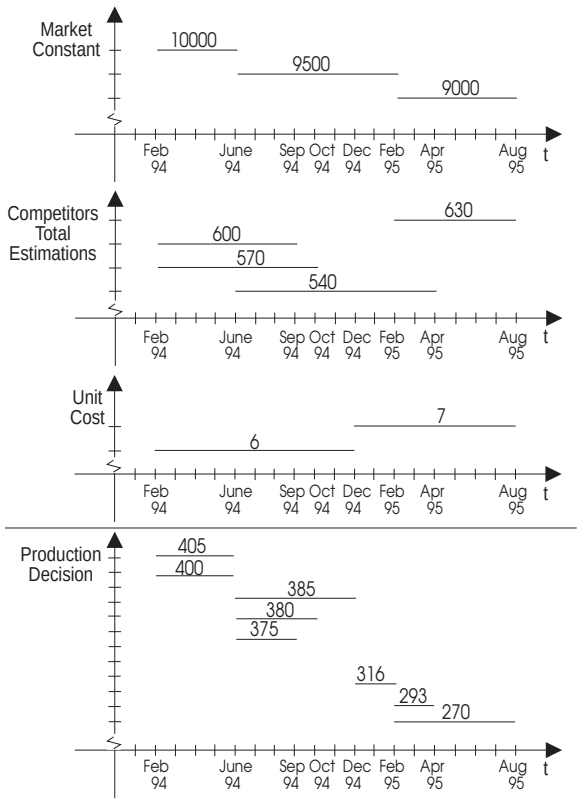

Figure 1: Exemplary data of Bilbo in the coffee manufacturers case study

valid during the time intervals as computed using all data-items that determine Production-Decision and shown in the bottom graph of Figure 1. For example, a Market-Constant of 10000, a Competitors-Total-Estimations of 600, and a Unit-Cost of 6, yield a Production-Decision of 400. Since during the interval [Jan 94, June 94), the value of Market-Constant is 10000, Competitors-Total-Estimations is either 600 or 570, and Unit-Cost is 6, Production-Decision in that interval is either 400 or 405.

As Figure 1 demonstrates, the number of values and their associated temporal intervals resulting from the computation of a single temporal data dependency may be very large. These values are a subset of the Cartesian product of all the possible values of each data-item. A naive approach would consider all the possible combinations in the Cartesian set ($3 \cdot 4 \cdot 2 = 24$ combinations in the example of Figure 1), yielding an algorithm with high time complexity. However, due to the bound temporal validity of values, usually only a small subset of the combinations in the Cartesian set should be considered. For example, in Figure 1 only

[1] We use a single month granularity, hence this interval is interpreted as all the days from February 1, 1994 to May 31, 1994 (June 1, 1994 is not included). A time interval is defined in [5] as "the time between two insatnces" and can be represented as either close or semi-open intervals.

8 combinations out of the 24 possible ones should be considered.

This work presents an algorithm that efficiently computes temporal data dependencies. Our approach for efficient evaluation of temporal data dependencies is based in part on previous works on computing temporal aggregates, including [8] and [6]. An aggregate function, such as selecting the minimum value of a set, is applied to a set of values (e.g. relations in the relational database model) and yields a scalar value. In temporal databases, the aggregate function is, in general, time dependent, i.e. the result of the aggregate function is applied to a set of values, each possibly having a different temporal validity. The calculation of temporal data dependencies is an extension of aggregate computing with *temporal grouping*, where the resulting values are grouped by time. To carry out such calculation, it is necessary to know which values have overlapping validity intervals, and to consider each value in its own validity interval.

The approach proposed in [8] first determines *constant intervals* as intervals within which there is no change in the data-item value. It then selects tuples that overlap each of these constant intervals and calculates the result. The work in [6] is based on a tree data structure for the time axis partition.

Extending these approaches to solve the problem of evaluating temporal data dependencies, we present a polynomial algorithm for efficient evaluation of temporal data dependencies with simultaneous values. The computation is not necessarily an aggregate operation that involves a single type of data-item with several values. Rather, it is a formula that may involve several types of data-items, each of which may consist of simultaneous values. The algorithm constructs a list sorted according to the time validity of data-items values, and then calculates the result as a function of the values in the list elements. Section 2 presents the algorithm for calculating temporal data dependencies with simultaneous values, while the properties of the algorithm are discussed in Section 3.

2 Evaluation of temporal data dependencies with simultaneous values

In this section we provide an outline of the algorithm for evaluating temporal data dependencies with data-items that consist of simultaneous values. The algorithm consists of two phases, namely generating a constant interval list and computing the value for each combination of each constant interval element, as follows.

The constant interval list is sorted by the starting time points of the constant intervals. The algorithm generates a partition of the valid time interval within which the temporal data dependency is to be determined. Each element of the constant interval list has a valid time τ, and it consists of all the values whose validity covers τ.

Initially, a constant interval element is generated with a valid time interval of $[t^s,\ t^e)$, where t^s and t^e are the start and end time points of the interval within which the temporal data dependency is to be evaluated, respectively. Each value is processed with respect to an interval that consists of its starting time point, as follows. Let $[t_s,\ t_e)$ be a valid time interval of a value val, and $[t^i_s,\ t^i_e)$ a constant time interval associated with a constant interval element ci_i. If $[t_s,\ t_e) \cap [t^i_s,\ t^i_e) \neq \emptyset$, then there are six possible relationships between $[t_s,\ t_e)$ and $[t^i_s,\ t^i_e)$, which are listed below along with the corresponding actions taken by the algorithm.

1. $t_s = t^i_s$ and $t_e < t^i_e$: replace ci_i with two constant interval elements, ci' with $[t_s, t_e)$ and ci'' with $[t_e,\ t^i_e)$. ci' and ci'' receive the values of ci_i, and val is added to ci'.
2. $t_s = t^i_s$ and $t_e = t^i_e$: add val to ci_i.
3. $t_s = t^i_s$ and $t_e > t^i_e$: add val to ci_i, and process val again with a valid time of $[t^i_e,\ t_e)$.
4. $t_s > t^i_s$ and $t_e < t^i_e$: replace ci_i with three constant interval elements, ci' with $[t^i_s,\ t_s)$, ci'' with $[t_s, t_e)$ and ci^* with $[t_e,\ t^i_e)$. ci', ci'' and ci^* receive the values of ci_i, and val is added to ci''.
5. $t_s > t^i_s$ and $t_e = t^i_e$: replace ci_i with two constant interval elements, ci' with $[t^i_s, t_s)$ and ci'' with $[t_s,\ t_e)$. ci' and ci'' receive the values of ci_i, and val is added to ci''.
6. $t_s > t^i_s$ and $t_e > t^i_e$: replace ci_i with two constant interval elements, ci' with $[t^i_s, t_s)$ and ci'' with $[t_s,\ t_e)$. ci' and ci'' receive the values of ci_i, and val is added to ci''. In addition, process val again with a valid time of $[t^i_e,\ t_e)$.

As an example of the activation of the first part of the algorithm, consider the data set of Figure 1, and assume that the interval within which the temporal data dependency is to be evaluated is [Feb 94, Aug 95). The initial element of the list would be $\langle$[Feb 94, Aug 95), Market-Constant=ϕ, Competitors-Total-Estimations=ϕ, Unit-Cost=$\phi\rangle$. The market-Constant values were processed first, then the Competitors-Total-Estimations values, and finally the Unit-Cost values. The full constant interval list is presented in Figure 2. To enhance comprehension, the figure presents all the elements that were generated throughout the process, in a form of a tree. Each node in the tree (except the root node) is an element of the list that was generated as a result of processing a value. A value at the bottom of a node represents the value whose processing resulted in splitting the node. The final Constant Interval List (CIL) is the set of all leaf nodes of the tree, represented in Figure 2 by bold rectangles. All other nodes were deleted during the process. The Production-Decision values, shown in Figure 1, are shown within the constant interval rectangles in Figure 2.

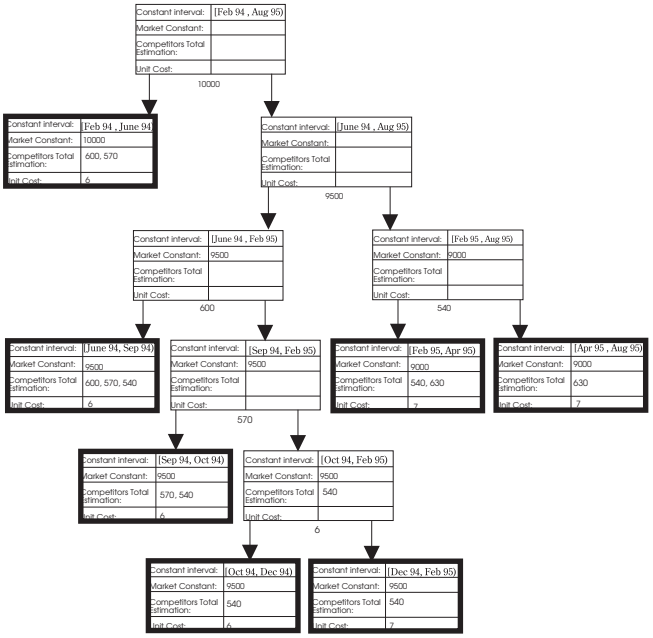

Figure 2: The constant interval list (CIL) generation process

A single constant interval element may consist of more than a single value for a data-item. For example, the constant interval element with the constant interval [Feb 94, June 94) in Figure 2 has two sets of values: $\langle 10000, 600, 6\rangle$, and $\langle 10000, 570, 6\rangle$. Each such set of values is a different *combination* for the calculation of the temporal data dependency, giving rise to a different value for the same interval.

The constant interval list may also be *over split* with respect to a combination, i.e. it may have consecutive constant interval elements with identical sets of values. For example, all the constant interval elements with the constant intervals [June 94, Sep 94), [Sep 94, Oct 94), and [Oct 94, Dec 94) consist of the combination $\langle 9500, 540, 6\rangle$. This is a result of several overlapping values of the same data-item. In this case, [June 94, Sep 94) and [Sep 94, Oct 94) are separate constant intervals, since the value 600 of Competitors-Total-Estimation is valid only during [June 94, Sep 94).

The second phase of the algorithm uses each of the possible value combinations in the constant interval elements to compute the new values. The combinations are scanned for each constant interval element, starting from the constant interval element with the minimal valid time. Subsequent constant interval elements are scanned to find identical combinations. If an identical combination is found, the valid time of the constant interval element is added to the valid time of the combination. This process is repeated until no more identical combinations can be found. For example, consider the example given in Figure 2. The combination $\langle 9500, 540, 6\rangle$ is selected from the constant interval element with the valid time of [June 94, Sep 94). The next constant interval element is scanned, and the same combination is found. Therefore, the valid time of the combination is set to be [June 94, Oct 94). The same combination is found again in the subsequent constant interval element, and the valid time of the combination is set to be [June 94, Dec 94). At this point, the process is terminated since the following constant interval element does not consist of this combination. The result of the second phase of the algorithm, applied on the data set of Figure 1 is the set of values of Production-Decision, which are also shown in Figure 1. After deciding on the appropriate interval, the values are used for calculating the derived value for that interval. For example, in the previous example, the derived value is calculated to be 385, valid during [June 94, Dec 94).

3 Algorithm properties

This section discusses the algorithm complexity (Section 3.1) and the correctness of the algorithm (Section 3.2).

3.1 Complexity

Let n be the number of processed values. A value with a valid time interval of $[t_s, t_e)$ can add two constant interval elements at the most, if for a constant interval element ci_i, $t_s > t_s^i$ and $t_e < t_e^i$, or if there are two constant interval elements ci_i and ci_j such that $t_s > t_s^i$ and $t_e < t_e^j$. Consequently, if the number of processed values is n, then the upper bound on the number of constant interval elements is $2n$.

For each value, the location of the first constant interval element to be processed is searched. This search is bounded by $\log_2(2n)$. In the worst case, a valid time of a value covers the valid times of all of the constant interval elements, resulting in $2n$ comparisons. Hence, the time complexity of the CIL generation phase is bounded by $\mathrm{O}(n(\log_2(2n) + 2n)) = \mathrm{O}(n^2)$.
The time complexity of the second phase of the algorithm is bound by $\mathrm{O}(m^3)$, where:

m is the number of all the valid combinations of a single value from all the data-items, i.e. all the combinations for which values have valid time overlapping. For example, in Figure 1, m=8 and the eight combinations are: $\langle 10000, 600, 6\rangle$, $\langle 10000, 570, 6\rangle$, $\langle 9500, 600, 6\rangle$, $\langle 9500, 570, 6\rangle$, $\langle 9500, 540, 6\rangle$, $\langle 9500, 540, 7\rangle$, $\langle 9000, 540, 7\rangle$, $\langle 9000, 630, 7\rangle$.

$2n$ is the maximal number of constant interval elements. for example, 18 is the maximal number of constant interval elements in Figure 1 since the number of state-elements is 9. However, The actual number of constant interval elements is 7, as shown in Figure 2.

$2mn$ is the maximal number of possible combinations in the list. The algorithm generates all of the list combinations ($2mn$). At each iteration of the algorithm, a single combination is processed. The worst case is when at each iteration all the remaining combinations are scanned. Thus, the worst case complexity is $\mathrm{O}(m^2 n)$. Since at each

constant interval element there is at least one combination, $m \geq n$. Therefore, the worst case complexity is bounded by $O(m^3)$.

From the discussion above we can conclude that the worst case complexity of the two phases of the algorithm is bounded by $O(m^3)$.

3.2 Correctness

Proposition 1 The partition of the time interval:
The entire set of constant interval elements constitute a partition of the evaluated interval.

The proof of Proposition 1 is done using induction on the number of constant interval elements.[2] At each iteration we verify that each of the six possible relationships between a valid time of a value and a constant interval element results in a new constant interval list that maintains the partition assertion. This proposition ensures the correctness of the list construction phase, where each value should allocate a single constant interval element.

Proposition 2 Algorithm correctness:
The algorithm generates a correct result.

Given a set of values and a temporal data dependency, a correct result ensures that for each combination of values with overlapping valid times there is a value which is the result of applying the temporal data dependency on this combination, and its valid time consists of the intersection of the overlapping valid times of the values in the combination. A simple algorithm, in which each combination of the Cartesian product is evaluated, can achieve a correct result but at a high computational cost. The proof of Proposition 2 shows that if a combination is not processed by the algorithm, then it should not be in the resulting set.

4 Conclusion and future research

We have proposed an algorithm for efficient evaluation of temporal data dependencies in temporal databases with simultaneous values. The algorithm consists of two phases, the first generates a constant interval list and the second computes the value for each combination of each constant interval element. A single constant interval element may consist of more than a single value for a data-item. The constant interval list may also be over split with respect to a combination, i.e. it may have consecutive constant interval elements with identical sets of values. The time complexity of the calculation algorithm is bound by $O(m^3)$, where m is the number of all the valid combinations of a single value from all the data-items. This result is less expensive, computation wise, than the result of scanning the Cartesian product ($O(\Pi_{i=1}^{n} m_i)$, where m_i is the number of values of the i-th data-item and n is the number of data-items involve in the calculation.

A prototype of a system that implements the algorithm exists on the basis of MAGIC 5.6 for DOS, under DOS 6.2. Further research is aimed at a more general form of temporal data dependencies, where the valid time is defined indirectly through constraints, or relative to other time points.

[2] Full definitions and proofs of propositions in this paper can be obtained via anonymous ftp to ftp.technion.ac.il under directory /usr/local/servers/ftp/pub/supported/ie. The file is called proofs.tex. It is produced using LaTeX. The proofs can also be obtained through the author's WWW home page, http://www.cs.toronto.edu/~avigal.

References

[1] A. Cournot. *Researches into the Mathematical Principles of the Theory of Wealth*. Macmillan, New York, N.Y., 1897.

[2] O. Etzion, A. Gal, and A. Segev. Temporal active databases. In *Proceedings of the International Workshop on an Infrastructure for Temporal Database*, June 1993.

[3] K.M. Van Hee, L.J. Somers, and M. Voorhoeve. A modeling environment for decision support systems. *Decision Support Systems*, 7:241–251, 1991.

[4] R. Hull and R. King. Semantic database modeling: Survey, application and research issues. *ACM Computing Surveys*, 19(3):201–260, Sep 1987.

[5] C.S. Jensen, J. Clifford, S.K. Gadia, A. Segev, and R.T. Snodgrass. A glossary of temporal database concepts. *ACM SIGMOD Record*, 21(3):35–43, 1992.

[6] N. Kline and R.T. Snodgrass. Computing temporal aggregates. In *Proceedings of the International Conference on Data Engineering*, pages 223–231, Mar 1995.

[7] J. Tirole. *The Theory of Industrial Organization*. the MIT press, 1989.

[8] P.A. Tuma. Implementing historical aggregates in TempIS. Master thesis. Wayne State University, Nov. 1992.

Appendix A: The production decision temporal data dependency

In this section we present the derivation of the temporal dependency graph given in Section 1. The following notation is used:
A ≡ Profit
B ≡ Revenue
C ≡ Cost
D ≡ Market-Price
E ≡ Fixed-Cost
F ≡ Unit-Cost
G ≡ Market-Constant
H ≡ Total-Quantity
I ≡ Competitors-Total-Estimation
X ≡ Production-Decision
We assume that Bilbo's strategy is to produce the amount that would maximize A, as follows.

$$
\begin{aligned}
A &= B - C = \\
&\quad X \cdot D - (E + X \cdot F) = \\
&\quad X \cdot \frac{G}{H} - (E + X \cdot F) = \\
&\quad X \cdot \frac{G}{X + I} - (E + X \cdot F)
\end{aligned}
$$

$$max\ (A) \Longrightarrow A' = 0$$

$$\Longrightarrow \frac{G \cdot (X + I) - X \cdot G}{(X + I)^2} - F = 0$$

$$\Longrightarrow \frac{G \cdot (X + I) - X \cdot G - F \cdot (X + I)^2}{(X + I)^2} = 0$$

$$\Longrightarrow F \cdot X^2 + 2 \cdot F \cdot I \cdot X + F \cdot I^2 - G \cdot I = 0$$

$$
\begin{aligned}
\Longrightarrow X &= \frac{-2 \cdot F \cdot I \pm \sqrt{4 \cdot F^2 \cdot I^2 - 4 \cdot F \cdot (F \cdot I^2 - G \cdot I)}}{2 \cdot F} = \\
&\quad - I \pm \sqrt{\frac{G \cdot I}{F}}
\end{aligned}
$$

X is non-negative.

$$\Longrightarrow X = max(\sqrt{\frac{G \cdot I}{F}} - I, 0)$$