Aggregates in generalized temporally indeterminate databases

Octavian Udrea Zoran Majkić V.S. Subrahmanian
University of Maryland
College Park, Maryland 20742 USA
{udrea,zoran,vs}@cs.umd.edu

Abstract

Dyreson and Snodgrass as well as Dekhtyar et. al. have provided a probabilistic model (as well as compelling example applications) for why there may be temporal indeterminacy in databases. In this paper, we first propose a formal model for aggregate computation in such databases when there is uncertainty not just in the temporal attribute, but also in the ordinary (non-temporal) attributes. We identify two types of aggregates: event correlated aggregates, and non event correlated aggregations, and provide efficient algorithms for both of them. We prove that our algorithms are correct, and we present experimental results showing that the algorithms work well in practice.

1 Introduction

Dyreson and Snodgrass [5] were the first to note that we are often not sure when an event may occur. For example, even though Fedex may tell us that a package will be delivered sometime today, if our *chronon* is "minute", then there is uncertainty about exactly at what time the package will be delivered. They go on to present a large set of convincing examples ranging from carbon-dating of archeological artifacts to scheduling applications where such uncertainty is the norm, not the exception, followed by an excellent framework for reasoning about such temporal indeterminacy. Later, Dekhtyar et. al. [3] built a rich temporal-probabilistic database algebra in which they could do away with many assumptions (e.g. independence) and extended the framework of Dyreson and Snodgrass.

Both the preceding works are *time centric* in the sense that uncertainty only exists in the temporal attributes of a relation. However, there are many applications where uncertainty can occur either in the temporal attributes or in the data (non-temporal) attributes, or in both. Our first major contribution is the concept of a "Generalized Probabilistic Temporal (GPT)" database that can handle both kinds of uncertainty. For example, almost every manufacturing company around today uses statistical models to estimate demand for a given product [13, chap.4] - independently of whether the product is high end (e.g. energy) or technology focused (e.g. digital cameras) or plain simple food (e.g. pasta). Such models estimate demand over time. They may estimate other parameters as well (e.g. price). Likewise, the entire agricultural sector is a poster child for temporal probabilistic data. Statistical models are used to predict how much of a particular crop (e.g. wheat) may be available, what the prices are likely to be, and so on. Such models are used by market analysts to make investments (e.g. into grain futures), by governments to decide what to import and when and in what quantity, and so forth [13]. In general, almost all applications involving economic principles are subject to uncertainty about supply, uncertainty about demand. Most of these uncertainties vary with time (e.g. demand for winter coats is usually small in the summer months).

Table 1 shows a sample data set about an energy market (virtually all US energy is sold by energy producers to energy distributors one day ahead using very complex statistical estimates of supply and demand). In this application, an energy producer is estimating demand (the Quantity field) for a given market as well as the Price (per unit of energy) to be charged for that quantity. For example, the quantity estimated for tomorrow in New York may be 5600 or 5700 units (with some probabilities) and the price per unit in the NY market may be be 115,600 or 115,700 per unit. Natural queries that corporate executives may be interested in include:

Q1: What is the expected demand in NY tomorrow?

Q2: What is my expected income tomorrow ?

Q3: On which day during the next 2 weeks period will I have the highest income?

All of these queries are aggregate queries.

Our second major contribution is the development of a declarative semantics for *aggregate queries* in GPT databases. Aggregates are of obvious interest in applications such as the above - a government might want to know

Table 1. Day-ahead energy market example

Event Id	Market	Price	Quantity	TP-case statement
1	NY	5600 [.6, .7] 5700 [.3, .4]	115600 [.5, .7] 115700 [.4, .6]	$\{\langle(10 \sim 14), (10 \sim 14), 0.4, 0.8, u\rangle\}$
2	Bos	5500 [.1, .1] 5600 [.2, .2]	104300 [.6, .8] 105000 [.3, .4]	$\{\langle(11 \sim 13), (11 \sim 13), 0.5, 0.9, u\rangle\}$
3	Wash	5650 [.5, .7] 5700 [.3, .5]	90500 [.5, .7] 92000 [.4, .6]	$\{\langle(10 \sim 12), (10 \sim 12), 0.4, 0.8, u\rangle\}$

the total expected wheat production in a given time period, while a manufacturer might want to know which market has the maximum profit margins.

Aggregate queries involving temporal probabilistic attributes fall into two general categories. Non-event correlated aggregates (NECA) are aggregates where all tuples in a GPT-relation are treated in "one pass." For example, queries (Q1) and (Q2) above fall into this category. In contrast, an event-correlated aggregate (ECA) is really an "aggregate over an aggregate." Query (Q3) falls into this category because we first need to find, for each day, the expected income for that day (this is an aggregate) and then we need to find the day that maximizes the expected income (which is an aggregate over the previously computed aggregates). Our third major contribution is the *development of algorithms* to efficiently compute NECA queries and ECA queries on GPT databases. In particular, we should mention that ECA queries can be speeded up by a "pre-aggregate" computation - when the database is updated, we need to update these pre-aggregates. We develop algorithms to do so.

Our fourth major contribution is a *prototype implementation* of our algorithms together with an experimental evaluation showing that our algorithms are very efficient. For instance, a SUM event-correlated aggregate can be computed in about 1.7 seconds over a database of 15,000 events in a disk-resident DB; when the number is increased to 500,000 disk-resident events, this can be done in about 5.1 seconds.

In this paper, we first extend the TP database model of Dekhtyar et. al. [3] to incorporate uncertainty in both the temporal and the data attributes – this is done in Section 2. We then develop a declarative definition of NECA and ECA aggregate queries in Section 4. Section 5 provides algorithms to compute the answers to NECA and ECA queries. Section 7 describes our prototype implementation.

2 GPT Database Model

2.1 Technical preliminaries

This section provides a brief overview of temporal probabilistic databases from [3]. We assume that $\tau = \{1, 2, \ldots, N\}$ for some integer N denotes the set of all legal time points — this time is discrete and modeled by the natural numbers. Throughout this paper, we will assume that τ is arbitrary but fixed. We assume the existence of a set of *time point variables* ranging over τ.

Definition 1 (Temporal constraint) *(i) If t_i is a time point variable, $\mathsf{op} \in \{<, \leq, =, \neq, \geq, >\}$, $v \in \tau$, then $(t_i \ \mathsf{op} \ v)$ is a temporal constraint.*

(ii) If $t_1, t_2 \in \tau$ and $t_1 \leq t_2$, then $(t_1 \sim t_2)$ is a temporal constraint (shorthand for $(t_1 \leq t \leq t_2)$; we abuse notation and write (t_1) instead of $(t_1 \sim t_1)$).

(iii) If C_1 and C_2 are temporal constraints then so are $(C_1 \wedge C_2)$, $(C_1 \vee C_2)$ and $(\neg C_1)$.

We use $\mathcal{S}_\tau$ to denote the set of all temporal constraints. The set $\mathrm{sol}(C)$ of solutions of a temporal constraint C is defined in the usual way. For example, $\mathrm{sol}((12 \sim 14) \vee (18 \sim 23)) = \{12, 13, 14, 18, 19, 20, 21, 22, 23\}$.

Definition 2 (Probability Distribution Function (PDF)) *A function $\wp : \mathcal{S}_\tau \times \tau \rightarrow [0, 1]$ is a* PDF *ifd $(\forall D \in \mathcal{S}_\tau)\,(\forall t \notin \mathrm{sol}(D))\,(\wp(D, t) = 0)$. Furthermore, $\wp$ is a* restricted PDF *ifd $(\forall D \in \mathcal{S}_\tau)\,(\sum_{t \in \mathrm{sol}(D)} \wp(D, t) \leq 1)$.*

This definition of a PDF is rich enough to capture almost all probability mass functions (e.g. uniform, geometric, binomial, Poisson, etc.)[12]. Furthermore, probability density functions can be approximated by PDFs via a process of quantization.

Definition 3 (TP-case) *A* TP-case *is a 5-tuple $\langle C, D, L, U, \delta\rangle$ where* (i) *C and D are temporal constraints,* (ii) *$\emptyset \subset \mathrm{sol}(C) \subseteq \mathrm{sol}(D)$,* (iii) *$0 \leq L \leq U \leq 1$, and* (iv) *$\delta$ is a restricted PDF.*

Consider the first event from the last column of Table 1 - the associated TP-case $\{\langle(10 \sim 14), (10 \sim 14), 0.4, 0.8, u\rangle\}$ says that the event with Event Id 1 will be true at some time between 10 and 14 with 40 to 80% probability. In general, C specifies the time points when an event is valid while D specifies the time points over which the PDF δ is applicable.

Table 2. Probabilistic flattening of a GPT-tuple

Event Id	Market	L_1, U_1	Price	L_2, U_2	Quantity	L_3, U_3	TP-case statement
1	NY	[1,1]	5600	[.6,.7]	115600	[.5,.7]	$\{\langle(10 \sim 14),(10 \sim 14),0.4,0.8,u\rangle\}$
1	NY	[1,1]	5600	[.6,.7]	115700	[.4,.6]	$\{\langle(10 \sim 14),(10 \sim 14),0.4,0.8,u\rangle\}$
1	NY	[1,1]	5700	[.3,.4]	115600	[.5,.7]	$\{\langle(10 \sim 14),(10 \sim 14),0.4,0.8,u\rangle\}$
1	NY	[1,1]	5700	[.3,.4]	115700	[.4,.6]	$\{\langle(10 \sim 14),(10 \sim 14),0.4,0.8,u\rangle\}$

Since $\mathrm{sol}(C) \subseteq \mathrm{sol}(D)$, it follows that δ assigns a probability to each time point $t \in \mathrm{sol}(C)$.

Definition 4 (TP-case statement) *A* TP-case statement Γ *is a set of TP-cases, where* $(\forall \gamma_i, \gamma_j \in \Gamma)((i \neq j) \rightarrow \mathrm{sol}(\gamma_i.C \wedge \gamma_j.C) = \emptyset)$. *We define* $\mathrm{sol}(\Gamma) = \bigcup\{\mathrm{sol}(\gamma_i.C) \mid \gamma_i \in \Gamma\}$.

The last column of Table 1 shows TP-cases for each of the three events in that relation.

Definition 5 (P-tuple) *Suppose* $[R] = (A_1, ..., A_k)$ *is a relation scheme in 1NF. A* P-tuple *over* $[R]$ *is a k-tuple* $pt = (\langle V_1, f_1\rangle, ..., \langle V_k, f_k\rangle)$ *where for all* $i \in [1, k]$, $V_i \subseteq dom(A_i)$ *and* f_i *is a function that maps each value in* V_i *to a probability interval (i.e., a closed subinterval of* $[0, 1]$*). For each attribute* A_i *we will call* V_i *the* value set *for that attribute.*

If we eliminate the last column of Table 1, we would have a P-relation.

2.2 GPT relations

In the following, we define a GPT tuple (or event) as a complex structure composed of an unique identifier, its temporal information Γ, and the set of probabilistic data (a P-tuple). For a relation R, we denote by $[R]$ the relation scheme of R.

Definition 6 (GPT tuples and relations) *A* general probabilistic temporal *tuple over the relation scheme* $[R] = (A_1, ..., A_k)$ *is a tuple* $gpt = (Id, pt, \Gamma)$*, where* Id *is the event's identifier,* $pt = (\langle V_1, f_1\rangle, ..., \langle V_k, f_k\rangle)$ *is a* P-tuple *and* Γ *is a* TP-case statement. *A* GPT-relation *over* R *is a finite set of* GPT-tuples *over* R.

Intuitively, a GPT-tuple is composed of a P-tuple and a TP-case statement, along with an event Id.

Example 1 The example given in Table 1 is a GPT-relation.

Definition 7 (FP-scheme) *Suppose* $[R] = (A_1, ..., A_k)$ *is a relation scheme in 1NF and L and U are probabilistic attributes where* $dom(L) = dom(U) = [0, 1]$. *We say that* $[fpR] = (A_1 : [L_1, U_1], ..., A_k : [L_k, U_k])$ *is an* FP-scheme *over* $[R]$ *with data attributes* $A_1, ..., A_k$ *and their probability intervals* $[L_i, U_i],\ i \in [1, k]$.

An FP-scheme resembles a simple relational scheme, but each attribute is "tied" to a probability interval. We now show how a given GPT relation can be "flattened" to an FP-scheme.

Definition 8 (Probabilistic flattening of GPT tuples) *For a given* GPT-tuple $gpt = (Id, pt, \Gamma)$*, where* $pt = (\langle V_1, f_1\rangle, ..., \langle V_k, f_k\rangle)$*, the* probabilistic flattened relation, $TP(gpt)$ *of gpt, is given by:* $TP(gpt) = \{(Id, \langle v_1, L_1, U_1\rangle, \ldots, \langle v_k, L_k, U_k\rangle, \Gamma) | (v_1, \ldots, v_k) \in V_1 \times \ldots \times V_k \wedge [L_i, U_i] = f_i(v_i))\}$. *The probabilistic flattening of a GPT-relation is the union of the probabilistic flattening of its GPT-tuples.*

Intuitively, we can flatten the P-tuple part of a GPT-tuple by taking the Cartesian product of all the V_i's. We then append the TP-case part of the P-tuple to each of the resulting tuples to get a flattened GPT-tuple.

Example 2 The probabilistic flattened relation (*TP relation*) of the event with $Id = 1$ in Table 1 is shown in Table 2.

Remark. Each tuple in the probabilistic flattening of an event can be regarded as a TP-tuple [3]. We will call such a tuple an *FPT-tuple* to emphasize that it was obtained through flattening and therefore its tuple data is an FP-scheme - meaning each data attribute is tied to a probability interval.

We now introduce the notion of a semi-annotated version of an FPT-tuple. This is done by taking each solution of a TP-case associated with that tuple and replacing the TP-case part of the FPT-tuple by that solution. In addition, the probabilities associated with that time point (solution of TP-case constraints) are added to the tuple.

Definition 9 (Semi annotation) *Let* $fpt = (Id, d, \Gamma)$ *be a* TP-tuple *over the relational scheme* $[R] = (A_1, ..., A_k)$ *with* $d = (\langle v_1, L_1, U_1\rangle, ..., \langle v_k, L_k, U_k\rangle)$ *as its data tuple. Then the* semi-annotated relation *for* fpt*, denoted* $SANN(fpt)$*, is defined as follows:* $SANN(fpt) = \{(Id, d, eTime, L_t, U_t) \mid \exists\ \gamma_i = \langle C_i, D_i, L_i, U_i, \delta_i\rangle \in \Gamma,\ s.t.\ (eTime \in sol(C_i)) \wedge ([L_t, U_t] = [L_i \cdot x, U_i \cdot x], x = \delta_i(D_i, eTime))\}$. *The semi annotated relation of an event* $gpt = (Id, pt, \Gamma)$*, denoted* $SANN(gpt)$*, is the union of all semi-annotated relations of its* FPT-tuples.

Table 3. Semi-annotated relation (partial)

Event Id	Market	L_1, U_1	Price	L_2, U_2	Quantity	L_3, U_3	eTime	L_t, U_t
1	NY	[1,1]	5600	[.6,.7]	115600	[.5,.7]	10	[.08,.016]
1	NY	[1,1]	5600	[.6,.7]	115600	[.5,.7]	11	[.08,.016]
1	NY	[1,1]	5600	[.6,.7]	115700	[.4,.6]	10	[.08,.016]
1	NY	[1,1]	5600	[.6,.7]	115700	[.4,.6]	11	[.08,.016]
1	NY	[1,1]	5700	[.3,.4]	115600	[.5,.7]	10	[.08,.016]
1	NY	[1,1]	5700	[.3,.4]	115600	[.5,.7]	11	[.08,.016]
1	NY	[1,1]	5700	[.3,.4]	115700	[.4,.6]	10	[.08,.016]
1	NY	[1,1]	5700	[.3,.4]	115700	[.4,.6]	11	[.08,.016]

Table 3 shows part of the semi-annotated version of the GPT-relation shown in Figure 1.

All operations of the temporal probabilistic algebra proposed in [3] can be extended to GPT-relations. We do not do this here as the focus of this paper is on aggregates which, to our knowledge, have never been defined for temporally indeterminate databases. The reader interested in the algebra can find it in an extended version of the paper at http://www.cs.umd.edu/users/udrea/GPTAggregates.pdf.

3 Probabilistic context

As we have seen in the previous section, the GPT model represents uncertainty both at the event and tuple data levels. As such, the computation of aggregates requires that we "combine" probability intervals. In other words, if $price_{NY} = 5600$ with probability in $[.6, .7]$ and $price_{Bos} = 5500$ with probability $[.1, .1]$, what is the probability that $price_{NY} + price_{Bos} = 11,100$? Clearly, this depends upon the relationship between prices in Boston and NY. If, for example, there are some power plants that can provide power to both Boston and NY, then there should be a correlation in price. On the other hand, if Boston and NY share no common power plants, then the prices are probably independent of each other.

Lakshmanan et. al. [7, 8] have proposed a very general notion of *conjunction and disjunction strategies*. Given two events e_1, e_2, each with probability intervals I_1, I_2 respectively, they define a conjunction strategy $\otimes$ to be an associative, commutative function that returns a probability interval $I_1 \otimes I_2$ for the conjunction $e_1 \wedge e_2$. Conjunction strategies are required to satisfy several other functions that we do not mention here. Disjunction strategies $\oplus$ do the same for disjunction. They show that these strategies are rich enough to express a wide variety of relationships between events such as independence, ignorance, positive and negative correlations, etc.

We will now define the concept of a *probabilistic context* that describes in precise terms how probability intervals for data (and temporal) attributes can be combined. For the rest of the paper, let $\mathcal{S}_C$ be the set of probabilistic conjunctive strategies and $\mathcal{S}_D$ the set of disjunctive strategies.

A probabilistic context (defined below) associates a conjunction and disjunction strategy with any set of attributes.

Definition 10 *(Probabilistic context) Let $[R]$ be a GPT relation scheme and $2^{[R]}$ be the power set of the set of attributes in $[R]$. A* probabilistic context *over $[R]$ denoted by $ctx(R)$ is a pair $\langle \otimes_{ctx}, \oplus_{ctx} \rangle$, where:*

(1) $\otimes_{ctx} : 2^{[R]} \rightarrow \mathcal{S}_C$ is a function that maps a set of attributes to a probabilistic conjunction strategy. Intuitively, for $W \in 2^{[R]}$ probability intervals would be combined using $\otimes_{ctx}(W)$ for any $|W|$-ary operator applied to values of the attributes in W. For singleton subsets $\{A\} \subseteq [R]$, $\otimes_{ctx}(\{A\})$ would be used in aggregate computations on A.

(2) $\oplus_{ctx} : 2^{[R]} \rightarrow \mathcal{S}_D$ is a function that maps a set of attributes to a probabilistic disjunctive strategy.

For the example described at the beginning of this section, using the independence conjunction strategy the result of $price_{NY} + price_{Bos}$ would be $\langle 11100, [.06, .07] \rangle$.

4 Temporal probabilistic aggregates declarative semantics

In this section, we define the formal semantics for the event and non-event correlated aggregation operators and show how these relate to aggregates on semi-annotated GPT relations. *We should note that even though the semantics for event correlated aggregation are based on their semi-annotated counterpart, our implementation computes such aggregates directly on GPT relations*. We start off with some simple examples that illustrate the differences between the two types of aggregation.

Example 3 Consider the relation shown in Table 1. We would like to answer the query *What is the total expected demand?* Let us assume that events are mutually independent (i.e. the independence conjunctive strategy is used for aggregation over the *Quantity* attribute).

Then the result of this query is the value $\langle V, f \rangle$, where $V = \{310400, 311900, 311100, \ldots\}$ and $f(310400) = [.15, .392]$, $f(311900) = [.12, .336]$, $f(311100) = [.06, .168]$, and so on. The TP-cases do not play a role in answering this query. When computing such aggregates, we only need ensure that different values from any single value set of the *Quantity* attribute are not aggregated. We will denote this type of aggregation **non-event correlated aggregation**.

Example 4 Let us consider the same relation, but with the query *When could the maximum demand occur?* One way to answer this query is via the following steps: (i) we find the set of time points T such that each time point in T is a solution to at least one TP-case in the relation; (ii) for each such time point $t \in T$, we add up the value of the *Quantity* attributes for all events e that have $t \in sol(e.\Gamma)$; (iii) choose the time point with the largest value for the aggregated attribute. Step (i) would give us $T = \{10, 11, 12, 13, 14\}$. For step (ii), part of the result is shown in Table 4 - again, assuming events are mutually independent. We call this type of aggregation **event correlated aggregation**.

***Note**: The reader may also notice that the query for this example actually includes two different aggregates - a* SUM *and a* MAX *operator; we use a combination of the two to illustrate why grouping by time points is relevant to the problem. In later sections, we will provide a much more efficient algorithm called **ECA-interval** that does not require examining each time point that is a solution of a TP-case.*

Table 4. Event correlated aggregation

eTime	SumOfQuantity
10	206100 [.25, .49] 207600 [.2, .42] 206200 [.2, .42] 207700 [.16, .36]
...	...
14	115600 [.5, .7] 115700 [.4, .6]

4.1 Aggregation operators

Definition 11 (Aggregation operator) *An aggregation operator agg is a function that takes a GPT-relation gpR and an attribute $A \in [gpR]$ as input and returns a GPT-relation gpR' containing **at most one** data attribute $A_{agg} = \langle V_{agg}, f_{agg} \rangle$. The values of A_{agg} are only over the values of attribute A.*

An aggregation operator produces a GPT-relation. For non-event correlated aggregation, the resulting relation would contain at most one tuple. For event correlated aggregation, gpR' may contain multiple tuples, each with its own event ID and set of TP-case statements. If the aggregation is performed over a data attribute in gpR, then gpR' would contain one data attribute with the aggregation result. However, if the aggregation is performed over the temporal attribute [1], gpR' will not contain any data attributes. Since aggregation on data attributes is a much more challenging problem, we will focus on such aggregates for the rest of the paper.

Due to the generality of aggregate operators as defined above, we believe an informal classification is in order. Aggregate operators over the temporal attribute are most likely to include *min, max* and *count*; sums or average computations over time points do not make sense. Data aggregate operators can be classified into **base** operators such as *sum*, *min*, *max*, *count* and **derived** aggregate operators such as *avg* or *stdev*. Derived operators are usually expressions involving base aggregates, data values and constants; for example, $avg = \frac{sum}{count}$. Computing such expressions is straightforward given the probabilistic context and the methods for combining probabilistic value sets that we have already informally described.

In this paper we focus on base aggregate operators such as *SUM*, *MIN*, etc. For notation purposes, for an aggregation operator agg we denote by:

- agg_r the corresponding relational aggregation operator.
- op_{agg} the corresponding arithmetic operator used to compute agg_r. For instance, if $agg = SUM$, $op_{agg} = +$. We expect that the binary op_{agg} operator is commutative and associative.
- op^*_{agg} the extension of op_{agg} to multisets of real numbers.

4.2 Non event correlated aggregation

This section will explore how non-event correlated aggregation can be performed on GPT tuples. As mentioned before, we will consider data tuple aggregates, as temporal data aggregates are straightforward problem.

Definition 12 (Non-event correlated GPT-aggregation) *Suppose R is a GPT-relation with probability context $ctx(R)$, $A \in [R]$ is an attribute and agg is an aggregation operator. For each tuple $t \in R$, we denote by $\langle V_t, f_t \rangle$, the value set and probability function for attribute A. Then, $agg(R, A) =_{def} \{(Id_*, \langle V, f \rangle, \Gamma)\}$, where:*

(i) Id_* is a new, dynamically created event ID.

[1] Example: What is the number of time points when the expected demand will exceed a certain threshold?

(ii) $V = \{op^*_{agg}(V') \mid V' \in \mathbf{X}_{t \in R} V_t\}$, *where* $\mathbf{X}$ *is the Cartesian product operator.*

(iii) *Let* $\otimes = \otimes_{ctx}(\{A\})$ *and* $\oplus = \oplus_{ctx}(\{A\})$. *Then* $\forall\ v \in V$, *the function* f *is defined by* $f(v) = \bigoplus \{\ f(v_1) \otimes \ldots \otimes f(v_N) |\ (v_1, \ldots, v_N) \in \mathbf{X}_{t \in R} V_t$ *and* $op^*_{agg}(\{v_1, \ldots, v_N\}) = v\ \}$, *where* $\bigoplus$ *is the extension of* $\oplus$ *to sets of probability intervals.*

(iv) $\forall \Gamma' \in [R].\Gamma, \forall t_i \in sol(\Gamma'),\ t_i \in sol(\Gamma)$.

This definition gives a method of computing non-event correlated probabilistic aggregates directly on GPT-relations. The reader may note that the intuition given in Example 3 is formalized here. The corresponding definition for non-event correlated aggregation for semi annotated relations is straightforward.

Table 5. Non event correlated aggregation

eTime	Sum(Price)
$\{\langle(10 \sim 14), (10 \sim 14), 0.4, 0.8, u\rangle,$	16750 [.03,.049]
$\langle(10 \sim 11), (10 \sim 11), 0.4, 0.8, u\rangle,$	16800 [.018,.035]
$\langle(10 \sim 12), (10 \sim 12), 0.4, 0.8, u\rangle\}$	16850 [.06,.126]
	16900 [.036,.09]
	16950 [.03,.056]
	17000 [.018,.04]

Example 5 For the GPT relation in Table 1, we choose $\otimes_{ctx}(\{Price\})$ to be the independence conjunction strategy and $\oplus_{ctx}(\{Price\})$ to be the ignorance disjunctive strategy defined in [7] as $[l_1, u_1] \oplus_{ig} [l_2, u_2] = [max(l_1, l_2), min(1, u_1 + u_2)]$. The result of the SUM non-event correlated aggregate over the $Price$ attribute can be found in Table 5.

4.3 Event correlated aggregation

We now define event correlated aggregation based on its semi-annotated counterpart. There are several advantages of defining aggregation for GPT relations in terms of its semi-annotated version: (i) correctness (i.e. commutativity with the semi-annotation function) is implied and (ii) the definition is less restrictive as to the form of the resulting GPT relation, which allows for more freedom in choosing an appropriate algorithm.

Definition 13 (World) *Let* R *be a GPT relation and let* $R' = SANN(R)$ *be its semi-annotated form. Let* $ctx(R)$ *be the probability context over* R, *let* $A_k \in [R]$ *be a data attribute in* $[R]$ *and let* $[L_k, U_k]$ *be the probability interval for* A_k *in* R'. *Suppose* $x \in \tau$ *is a time point. A* world *for* x, R, A_k *is any* subset w *of the set* $\pi_{(Id, eTime, A_k, L_k, U_k)}(\sigma_{(eTime = x)}(R'))$ *such that* $\forall\ t_1, t_2 \in w, t_1.Id \neq t_2.Id$ [2]. *A* maximal world *for a time point* x *is a world* w' *such that there does not exist another world* w'' *for* x *such that* $w' \subset w''$.

Intuitively, a maximal world for a time point represents a possible combination of values from the value sets for A_k. This is conceptually similar to an element in the Cartesian product present in Definition 12. Furthermore, a maximal world contains tuples for all possible events that contain such values. For a time point x, a relation R and an attribute $A \in [R]$, we denote by $\mathcal{W}_x(R, A)$ the set of maximal worlds over R and A w.r.t. x. Furthermore, given an aggregate operator agg and its relational counterpart agg_r, we denote by $agg_r(\mathcal{W}_x(R, A)) = \{y | \exists\ w \in \mathcal{W}_x(R, A)\ s.t.\ agg_r(w, A) = y\}$. In short, $agg_r(\mathcal{W}_x(R, A))$ is the set of all possible values obtainable through relational aggregation through agg_r over any maximal world w and the attribute A. Similarly, we define $agg_r^{-1}(y, \mathcal{W}_x(R, A)) = \{w \in \mathcal{W}_x(R, A) | agg_r(w, A) = y\}$.

Definition 14 (Semi annotated aggregation) *Let* R *be a GPT relation and let* $R' = SANN(R)$ *be its semi-annotated form. Let* $ctx(R)$ *be the probability context over* R, *let* $A_k \in [R]$ *be a data attribute in* $[R]$ *and let* $[L_k, U_k] \in [R']$ *be its corresponding probabilistic attribute in the semi-annotated form. Let* agg *be an aggregation operator and* agg_r *the corresponding relational aggregate. We define the semi-annotated version of* agg *as* $agg_{FL}(R', A_k) = \{t | t = (Id_*, eTime, A_k^{agg}, [L_k, U_k]^{agg})\}$, *where* $\forall\ x \in R'.eTime, \forall\ y \in agg_r(\mathcal{W}_x(R, A)), \exists!\ t \in agg_{FL}(R', A_k)$ *such that:*

(1) $t.Id_*$ *is a new, dynamically generated event identifier.*

(2) $t.eTime = x$ *and* $t.A_k^{agg} = y$.

(3) *Let* $\otimes = \otimes_{ctx}(\{A_k\})$ *and* $\oplus = \oplus_{ctx}(\{A_k\})$. *Let* $\bigoplus$ *be the extension of* $\oplus$ *to a multiset of probability intervals and* $\bigotimes$ *the similar extension of* $\otimes$. *For a world* $w \in \mathcal{W}_x(R, A_k)$, *let* $w.[L_k, U_k]$ *be the multiset of probability intervals that appear in that world. Let* $\mathcal{I}$ *be the multiset* $\{[L, U] | \exists\ w \in agg_r^{-1}(y, \mathcal{W}_x(R, A))\ s.t.\ [L, U] \in w.[L_k, U_k]\}$. *Then* $t.[L_k, U_k]^{agg} = \bigoplus(\mathcal{I})$.

This definition formalizes the intuition given in Example 4 for the semi-annotated version of the GPT relation. In short, we perform all possible combinations of values in the value set of A_k, while grouping them by time point and insuring that two different values from the same value set do not participate in the same aggregate value. We can now define event correlated aggregation for GPT relations. We denote

[2] Note that w is a set, therefore duplicate tuples for the values in A_k are ignored.

by π_{-Id} the projection operation that select all attributes of a GPT relation except the event Id.

Definition 15 (Event correlated aggregation) *Let R be a GPT relation, let $A \in [R]$ be a data attribute in $[R]$. Let $ctx(R)$ be the probabilistic context over R, let agg be an aggregate operator and let agg_{FL} be its semi-annotated counterpart. The result of the application of agg to attribute A in $[R]$ is a GPT relation that satisfies the following: $\pi_{-Id}(SANN(agg(R, A))) = \pi_{-Id}(agg_{FL}(SANN(R), A))$.*

5 Algorithms for computing aggregates

Aggregate computation in the GPT model poses a series of new challenges due to the presence of uncertainty both in temporal and regular attributes.

Problem 1. NECA aggregation is directly defined on GPT relations. ECA aggregation on the other hand is defined w.r.t. the semi-annotated version of a GPT-relation. However, semi-annotation involves an significant space blowup which we would like to avoid. As such, our algorithms (both for NECA and ECA aggregation) work directly with GPT relations.

Problem 2. Let us consider a GPT relation R and an attribute $A \in [R]$ such that $\forall\, t \in R$, $t.A = \langle V_t, f_t \rangle$, $|V| \leq c$, where c is an arbitrary constant. Let agg be an aggregation operator. According to Definition 12 (for NECA) and Definitions 14 and 15, in the worst case scenario the space complexity of the result would be $\mathcal{O}(c^N)$, where $N = |R|$.

It is obvious that in the case of Problem 2, an exponential complexity (both in space and time) is unacceptable. Furthermore, it is unlikely that a result with a data size exponential in the size of the initial relation would be of any use to an end user. As such, we see two possible scenarios: either the size of the input relation is limited by a selection query (as is usually the case with aggregates on relational data) or aggregate computations are run in sequence, as is the case in Example 4. For the sake of generality, we will assume the existence of a *restrict* method [3] that restricts the set of values V_t that are to be considered for further computation for any tuple t at any intermediate step. We assume the existence of such a method for both event and non-event correlated aggregations.

5.1 NECA algorithm

We now present the *NECA* algorithm for computing non-event correlated aggregates on GPT-relations. The algorithm takes advantage of our earlier assumption that op_{agg} is associative and commutative. Likewise, Lakshmanan et. al. [7] assume that all conjunctive and disjunctive strategies are associative and commutative - something we assume as well.

Algorithm: NECA(R,A,agg,$ctx(R)$)
Input: GPT relation R, attribute $A \in [R]$,aggregation operator agg, probabilistic context $ctx(R)$.
Output: GPT relation R_{agg} representing the result of non-event correlated aggregation.
Notation: $\otimes = \otimes_{ctx}(\{A\})$, $\oplus = \oplus_{ctx}(\{A\})$;

```
1.  Γ ← ∅;
2.  V ← ∅;
3.  f ← null;
4.  for all t ∈ R do
5.    Γ ← Γ ∪ t.Γ;
6.    if V = ∅ then
7.      ⟨V, f⟩ ← t.A;
8.    else
9.      V' ← V X t.A.V;
10.     V ← ∅;
12.     for all (v1, v2) ∈ V' do
13.       V ← V ∪ {v1 op_agg v2};
14.     V ← restrict(V);
15.     for all v ∈ V do
16.       f(v) ← 0;
17.       for all (v1, v2) ∈ V' s.t. v1 op_agg v2 = v do
18.         f(v) ← f(v) ⊕ (f(v1) ⊗ t.A.f(v2));
19.     endfor
20.   endif
21. endfor
22. R_agg ← {(Id*, Γ, ⟨V, f⟩)};
23. return R_agg;
```

The algorithm performs an incremental computation by analyzing one tuple in the input relation at a time and maintaining an intermediate result. The *restrict* method is used at each step to restrict the number of values from the intermediate value set that are to be considered further during the computation. As an example if the query planner can determine that a *MIN* aggregate is to be executed on this result, the *restrict* method would only select the smallest value computed at each step.

Theorem 1 (NECA correctness) *The* NECA *algorithm terminates and the resulting GPT relation is correct w.r.t. Definition 12 when $restrict$ is the identity function.*

Justification. Termination is straightforward: the loop on line 4 is on the tuples in R, which is not modified by the algorithm; the loops on lines 12 and 15 are on finite value sets and the loop on line 17 is on a Cartesian product of these sets. Given that $restrict$ is the identity function, the algorithm computes the set V in Definition 12 on a step-by-step basis on line 9. Furthermore, the computation of $f(v)$ for each value v in the resulting value set (line 18) is correct since all probabilistic conjunctive and disjunctive strategies

[3] If aggregates are run in sequence, then the *restrict* method can be easily provided by the query planner. For Example 4, we would only look at the maximum value from any value set V_t for a tuple t, knowing that a *MAX* query is to be applied next. If the selection query case holds, then the *restrict* method can simply let all values in V_t go through.

are required to be commutative and associative by [7]. The condition on line 17 accounts for item (iii) in Definition 12.

Let us assume that the set returned by the *restrict* method is bounded by $\mathcal{O}(r)$ and the size of the value sets for attribute A are bounded by an arbitrary constant c. For each tuple, at most $\mathcal{O}(r)$ values held in the intermediate result are combined with at most $\mathcal{O}(c)$ values from the new tuple. This operation is performed for each tuple in the input relation, therefore the complexity of the *NECA* algorithm is $\mathcal{O}(n) \cdot \mathcal{O}(c \cdot r)$

5.2 ECA algorithms

In this section, we present two algorithms to compute ECA aggregates. We remind the reader that even thought the declarative semantic in Definition 15 is based on the semi-annotated corresponding aggregate, our methods avoid the space blowup required by semi-annotation and compute aggregates directly on GPT relations. The first algorithm, *ECA-timepoint*, is a simple method for computing event correlated aggregation. The second algorithm *ECA-Interval*, is far more efficient.

ECA-timepoint is based on the *NECA* algorithm. A simple probabilistic flattening of the result of the *ECA-timepoint* yields the corresponding semi-annotated aggregate. However, as our experiments will show, the *ECA-timepoint* algorithm is resource consuming both in terms of execution time and space, as the result is dependent on the granularity of the temporal information.

Algorithm: ECA-timepoint(R,A,agg,$ctx(R)$)
Input: GPT relation R, attribute $A \in [R]$,aggregation operator agg, probabilistic context $ctx(R)$.
Output: GPT relation R_{agg} representing the result of event correlated aggregation.
Notation: (t, L, U) is the TP-case statement that only contains t as a solution.
1. $R_{agg} \leftarrow \emptyset$;
2. **for** all $t \in \tau$ **do**
3. $R' \leftarrow \sigma_{t \in sol(\Gamma)}(R)$;
4. $R'' \leftarrow NECA(R', A, agg, ctx(R))$;
5. $[L, U] \leftarrow [0, 0]$;
6. **for** all $u \in R'$ and $\gamma \in u.\Gamma$ **do**
7. **if** $t \in sol(\gamma)$ **then**
8. $[L, U] \leftarrow [L, U] \oplus_{ctx} (\{\Gamma\})\ [\gamma.L \cdot \delta(\gamma.D, t), \gamma.U \cdot \delta(\gamma.D, t)]$;
9. **endfor**
10. $Ragg \leftarrow R_{agg} \cup \{(Id_*, (t, L, U), R''.A_{agg}\}$;
11. **endfor**
12. **return** R_{agg};

The correctness of the *ECA-timepoint* algorithm follows directly from Definition 15 as the probabilistic flattening of the resulting relation yields the same result as semi-annotated aggregation. The complexity of the algorithm is clearly $\mathcal{O}(|\tau|) \cdot \mathcal{O}(NECA)$, since the for loop on line 6 is run only for events that have t as a solution, while the complexity of the *NECA* algorithm is higher.

The *ECA-timepoint* algorithm has several disadvantages: (i) the storage space for the result is highly dependent on the granularity of time and (ii) a lot of tuples in the result may be identical except for the time point. For instance, the computation of one aggregate value may be repeated multiple times if several time points correspond to the same GPT-tuples.

We try to address the above issues with the more advanced *ECA-interval* algorithm which makes use of interval constraints to perform the computation of each aggregate value only once. The *ECA-interval* algorithm uses a *pre-aggregation* relation that contains partial information to be included in the result. Simply put, a pre-aggregate is a GPT relation that contains temporal data (Γ) and an IDS attribute. Each tuple in the pre-aggregate corresponds to a tuple in the result **independent** of the aggregate operator used. The result will replicate the temporal attribute and aggregate all events whose IDs are in the IDS set[4].

Table 6. Pre-aggregate relation

Γ	**IDS**
$\{\langle(10), .2, .38\rangle\}$	$\{1, 3\}$
$\{\langle(11), .34, .57\rangle, \langle(12), .34, .57\rangle\}$	$\{1, 2, 3\}$
$\{\langle(13), .23, .41\rangle\}$	$\{1, 2\}$
$\{\langle(14), .08, .16\rangle\}$	$\{1\}$

Example 6 For the GPT relation in Table 1, we choose $\oplus_{ctx}(\Gamma)$ to be the independence disjunction strategy. The pre-aggregate relation for this case is show in Table 6 [5].

Theorem 2 *The* ECA-interval *algorithm terminates and produces a correct result w.r.t. Definition 15 when* $restrict$ *is the identity function.*

Justification. The *pre_aggregate* method terminates because at each step, the "overlapping" of TP-case statements in the Pre relation is reduced. Furthermore, the algorithm itself contains a single loop on the finite set of tuples in the Pre relation. The correctness follows directly from the way the pre-aggregate relation is constructed.

We should note that the worst-case complexity of the *ECA-interval* algorithm is still directly dependent on $|\tau|$. However, our experiments show that even for randomly generated data, the *ECA-interval* algorithm greatly outperforms the time point version. Furthermore, the space requirements for storing the result are also decreased. A note-

[4] In a non-event correlated manner.

[5] Probabilities were restricted to 2 figures after the decimal point.

method pre_aggregate(R,$ctx(R)$)
Input: GPT relation R, probabilistic context $ctx(R)$.
Output: GPT relation Pre with attributes Γ and IDS. The IDS attribute contains a set of event IDs that correspond to the TP-case statements in the tuple.
Notation: $\otimes = \otimes_{ctx}(\{\Gamma\})$; $\oplus = \oplus_{ctx}(\{\Gamma\})$; (t, L, U) is the TP-case statement that only contains t as a solution.
Comments: The *computeLU* method for a time point used here follows the computation of L,U for a time point in line 8 of the *ECA-timepoint* algorithm [a].

1. $Pre \leftarrow \emptyset$;
2. **for** all $t \in R$ **do**
3. $\Gamma \leftarrow t.\Gamma$;
4. **for** all $u \in Pre$ **do**
5. **if** $sol(\Gamma) \cap sol(u.\Gamma) \neq \emptyset$ **then**
6. **for** all $\gamma_1 \in \Gamma, \gamma_2 \in u.\Gamma$ s.t. $sol(\gamma_1) \cap sol(\gamma_2) \neq \emptyset$ **do**
7. $u.\Gamma \leftarrow u.\Gamma - \{\gamma_2\}$;
8. $u.\Gamma \leftarrow u.\Gamma \cup \{\langle x, computeLU(x)\rangle | x \in sol(\gamma_2) - sol(\gamma_1)\}$;
9. $\Gamma \leftarrow \Gamma - \{\gamma_1\}$;
10. $\Gamma \leftarrow \Gamma \cup \{\langle x, computeLU(x)\rangle | x \in sol(\gamma_1) - sol(\gamma_2)\}$;
11. $\Gamma' \leftarrow \{\langle x, computeLU(x)\rangle | x \in sol(\gamma_1) \cap sol(\gamma_2)\}$;
12. $Pre \leftarrow Pre \cup \{\Gamma', u.IDS \cup \{t.Id\}\}$;
13. **goto** 4;
14. **endfor**
15. **endif**
16. **endfor**
17. $Pre \leftarrow Pre \cup \{(\Gamma, \{t.Id\})\}$;
18. **endfor**
19. **return** Pre;

[a] In cases where the same time point appears in multiple single time point constraints in the same Γ, a *compaction* method can be used to reduce data sizes. Compaction functions have been discussed in [3].

worthy feature of this algorithm is the possibility of running computations for each tuple in the result in parallel[6].

6 Pre-aggregate maintenance

As was pointed out in the previous section and as we will later show experimentally, the efficiency of the *ECA-interval* algorithm is based on the computation of the pre-aggregate relation. Therefore, in this section we will address the problem of pre-aggregate maintenance for insert and delete operations [7]. We do not directly address the problem of maintaining aggregate views on GPT-relations; however, since the tuples in the aggregate relation have a one-to-one correspondence to the tuples in the pre-aggregate, view maintenance can be achieved through extensions of the algorithms described here.

The *PAM-Delete* algorithm performs pre-aggregate maintenance for delete operations. We identify all records

[6] Although theoretically possible for the *ECA-timepoint* algorithm, the parallelism would be reduced because of the number of tuples in the result.

[7] For reasons of space, we will treat an update as a delete/insert combination. However, there is a straightforward optimization of this approach that directly handles data modification operations.

Algorithm: ECA-interval(R,A,agg,$ctx(R)$)
Input: GPT relation R, attribute $A \in [R]$,aggregation operator agg, probabilistic context $ctx(R)$.
Output: GPT relation R_{agg} representing the result of event correlated aggregation.

1. $Pre \leftarrow pre\_aggregate(R, ctx(R))$;
2. **for** all $t \in Pre$ **do**
3. $R' \leftarrow NECA(\sigma_{Id \in t.IDS}(R), A, agg, ctx(R))$;
4. $R_{agg} \leftarrow R_{agg} \cup \{Id_*, t.\Gamma, R'.A_{agg}\}$;
5. **endfor**
6. **return** R_{agg}.

in the pre-aggregate relation that contain the Id of the deleted event e. If in the corresponding temporal attribute there are any single time point constraints $\langle(t), L, U\rangle$ such that $t \in sol(e.\Gamma)$, the probability bounds L and U are recomputed only if t is also a solution for other events appearing in the same pre-aggregate tuple. Otherwise, the constraint is simply deleted. We should note that according to the *ECA-interval* algorithm, for every TP-case γ that appears in the pre-aggregate relation: (i) if $\gamma = \langle(t), L, U\rangle$, then t can be a solution of multiple events in the original relation; (ii) if γ contains a complex constraint, then γ appears in exactly one TP-case statement in the original relation.

Algorithm PAM-Delete(e, PR)
Input: The event to be deleted e and the pre-aggregate relation PR.
Output: The new pre-aggregate relation.

1. **for** all $t = \langle\Gamma, IDS\rangle \in PR$ **do**
2. **if** $e.Id \in t.IDS$ **then**
3. **for** all $\gamma \in t.\Gamma$ **do**
4. **if** $\gamma = \langle(t), L, U\rangle$ **then**
5. **if** $x \in sol(e.\Gamma)$ **then**
6. $t.\Gamma \leftarrow t.\Gamma - \{\gamma\}$;
7. $t.\Gamma \leftarrow t.\Gamma \cup \{\langle(x), computeLU(x)\rangle\}$;
8. **endif**
9. **else if** $\gamma \in e.\Gamma$ **then**
10. $t.\Gamma \leftarrow t.\Gamma - \{\gamma\}$;
11. **endif**
12. **endfor**
13. $t.IDS \leftarrow t.IDS - \{e.Id\}$;
14. **endif**
15. **endfor**
16. **return** PR;

The complexity of the algorithm depends on the size of the pre-aggregate relation (line 1), but more importantly on the overlap [8] between events in the original relation. This is due to the fact that one of the time consuming operations is the recomputation of probability bounds for single time point constraints (line 7). We will later show experimentally that the overlap also influences running time and storage space for the algorithms described.

The *PAM-insert* algorithm - which addresses insert

[8] The number of events that have the same time point(s) as a solution

operations - is more complex. In this case, we will have to determine which events from the original relation "overlap" the new event w.r.t. the solution to the temporal constraints; for each event whose solution intersects that of the inserted tuple, the corresponding tuples in the pre-aggregate relation must be modified accordingly. Either these tuples can incorporate the temporal information of the inserted event or they must be "split" to accommodate the insert operation.

Algorithm PAM-insert(e, PR)
Input: Event $e = \langle Id, \Gamma, dt\rangle$ to be inserted, pre-aggregate relation PR.
Output: The new pre-aggregate relation.
Comments: Q is a FIFO queue.
1. $Q \leftarrow enqueue(e)$;;
2. **while** $Q \neq \emptyset$ **do**
3. $ev \leftarrow dequeue(Q)$;
4. **for** all $t = \langle \Gamma, IDS\rangle \in PR$ **do**
5. **if** $sol(t.\Gamma) \cap sol(ev.\Gamma) \neq \emptyset$ **then**
6. $\Gamma' \leftarrow \emptyset$;
7. $\Gamma'' \leftarrow \emptyset$;
8. **for** all $\gamma_1 \in t.\Gamma, \gamma_2 \in ev.\Gamma$ **do**
9. **if** $sol(\gamma_1) \cap sol(\gamma_2) \neq \emptyset$ **then**
10. $t.\Gamma \leftarrow t.\Gamma - \{\gamma_1\}$;
11. $t.\Gamma \leftarrow t.\Gamma \cup \{\langle x, computeLU(x)\rangle | x \in sol(\gamma_1) - sol(\gamma_2)\}$;
12. $\Gamma' \leftarrow ev.\Gamma - \{\gamma_2\}$;
13. $\Gamma' \leftarrow \Gamma' \cup \{\langle x, computeLU(x)\rangle | x \in sol(\gamma_2) - sol(\gamma_1)\}$;
14. $\Gamma'' \leftarrow \{\langle x, computeLU(x)\rangle | x \in sol(\gamma_1) \cap sol(\gamma_2)\}$;
15. **endif**
16. **endfor**
17. $PR \leftarrow PR \cup \{\langle \Gamma'', t.IDS \cup \{ev.Id\}\rangle\}$;
18. $Q \leftarrow enqueue(\langle ev.Id, \Gamma', ev.dt\rangle)$;
19. **endif**
20. **endfor**
21. **endwhile**
22. **return** PR;

The termination of the algorithm follows from the fact that the size of the overlap between the inserted event and other events in the original relation decreases at each iteration over the queue. The queue maintains that part of the temporal information for the event to be inserted whose solution does not intersect the temporal information from the current pre-aggregate tuple (lines 13, 18). Thus the algorithm produces a valid pre-aggregate relation, in that any two tuples in PR have disjoint solutions for the temporal part.

7 Experimental results

We have developed a 6570 line Java implementation of the algorithms described in this paper. Our experiments were run on a Pentium 4 3.2Ghz machine with 1GB of RAM. The GPT database was built on top of PostgreSQL 8.0. The series of experiments described in this section were run on synthetically generated data with between 14,500 events and 500,000 events, all data being stored on disk. During the experiments we have identified several factors that impact the performance and storage space requirements for our algorithms. Among these were: (i) the data size; (ii) the type of aggregate query - SUM queries are much more expensive than MIN, MAX, $COUNT$; this also involves the $restrict$ method bounds; (iii) the "overlapping" factor l - which measures the degree of temporal overlap between events. Strictly speaking, l was computed as an average on the multiset $\{x | \exists\ t \in \tau\ s.t.\ \exists\ x\ different\ events\ with\ t\ as\ a\ solution\}$.

Impact of Size. We measured the impact of the input relation size on the running time of queries. We fixed the $restrict$ method to select only the maximum value from each value set, similarly to Example 4. The overlapping factor was $l \approx 8.5$. We applied a SUM aggregate both in non-event and event correlated manner, plotting the running time for each of the three algorithms. The time taken to construct the pre-aggregate relation was measured independently. Figure 1 shows the experimental results [9]. We can easily see that the non-event correlated aggregation is only slightly faster than the *ECA-interval* algorithm. The reason for this is that in the *ECA-interval* case, once the pre-aggregated relation is computed, the *NECA* algorithm is applied to small subsets of the tuples in the input relation, whereas in the *NECA* case, the algorithm is applied to the whole relation. We can also see that the *ECA-interval* outperforms the *ECA-timepoint algorithm*. Moreover, the parallelized version of the *ECA-interval* algorithm using 40 worker threads is as efficient as the semi-annotated version of the aggregation.

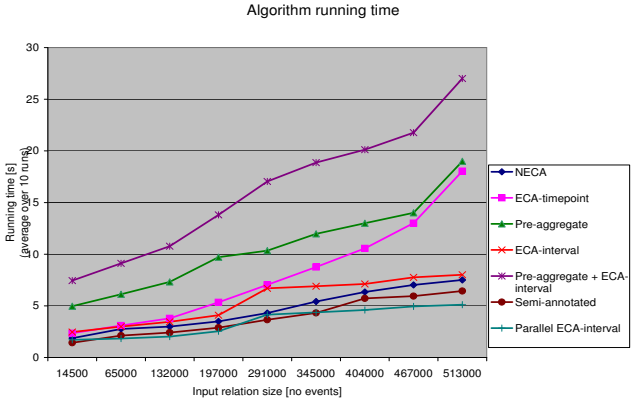

Figure 1. Algorithm running time

Impact of Overlapping Factor. We fixed the input relation size to 255000 events and we measured the impact of the overlapping factor on the relative performance of the

[9] Images are best viewed in color.

ECA-interval and *ECA-timepoint* algorithms. The *NECA* algorithm is not affected by these experiments, as it does not consider temporal information in the process of computing an aggregate value. The $restrict$ method was the same as mentioned above. The results in Figure 2 show that once the overlapping factor starts to increase, the *ECA-interval* algorithm slowly tends toward the same running time as *ECA-timepoint* due to the increased number of single time point constraints in the pre-aggregation relation. However, as the overlapping factor increases over a certain threshold, more events will correspond to the same tuple in the result - since τ is finite, it can only mean the size of the overlapping intervals increases, meaning *ECA-interval* is much more efficient than *ECA-timepoint*.

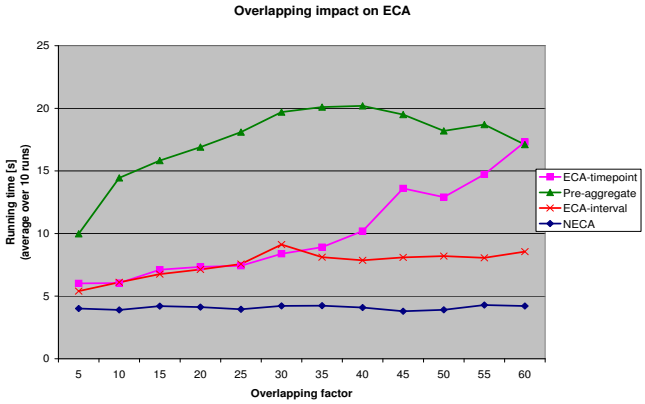

Figure 2. Overlap impact on performance

Size of output. We measured the storage space needed for the results of the aggregation. The critical factor here is again the overlapping factor. The *NECA* algorithm only produces one tuple, and thus is storage space is minimal with a reasonable $restrict$ method - the only variations are due to the representation of a compact union of all temporal information. The overlapping factor was fixed to the same value as in the first set of experiments. The *ECA-timepoint* is the most inefficient from this point of view, since it stores one tuple for each time point, whereas the *ECA-interval* algorithm minimizes the storage space for event correlated aggregation. The results can be seen in Figure 3.

Performance of PAM-delete *and* PAM-insert. We assessed the performance of these algorithms in terms of delete/insert operations handled per second. We varied both the input relation size and the overlapping factor, as both influence the complexity of these algorithms. As expected, *PAM-delete* is reasonably fast even for large datasets, as detailed in Figure 4. Moreover, it has an almost linear dependence both on the size of the input relation and on the overlapping factor.

Figure 5 shows the performance results for the *PAM-*

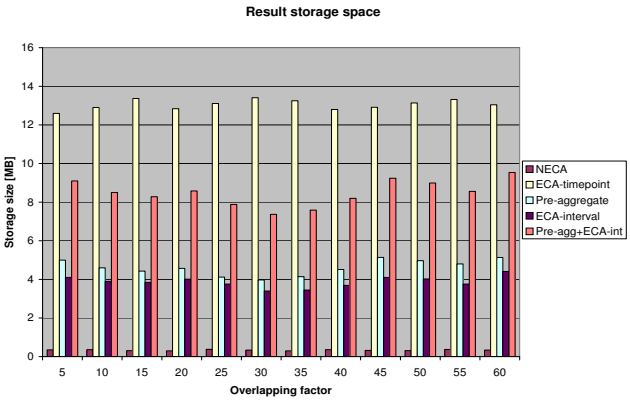

Figure 3. Overlap impact on storage space

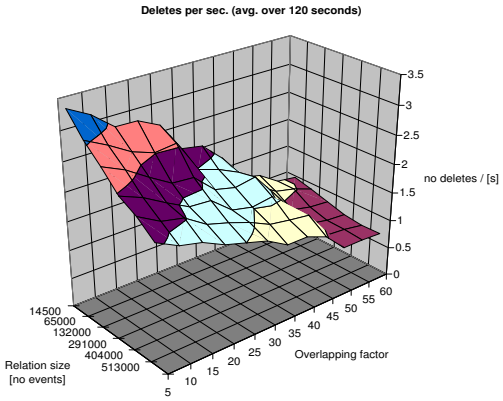

Figure 4. PAM-Delete performance

insert algorithm. As expected, the number of inserts per second is lower than that of the delete operations, due to the complexity of the algorithm. However, we notice *PAM-insert* does not depend as much on the overlap factor as *PAM-delete*. The reason for this behavior is that the effect of the input relation size is much higher: as the number of events increases, the likelihood that the inserted event overlaps with more tuples in the pre-aggregate also increases. The effect of the overlap factor is smaller in comparison.

8 Related work and conclusions

The business world is full of economic models that are full of uncertainty about supply (of a resource), demand (for the resource). Often times, supply and demand have a temporal aspect to them - supply and demand for sweaters is far greater in the winter months than in the summer. In this paper, we have used a real-world energy model [14] that we have worked on to motivate the need for reasoning about uncertainty in domains where time plays a role.

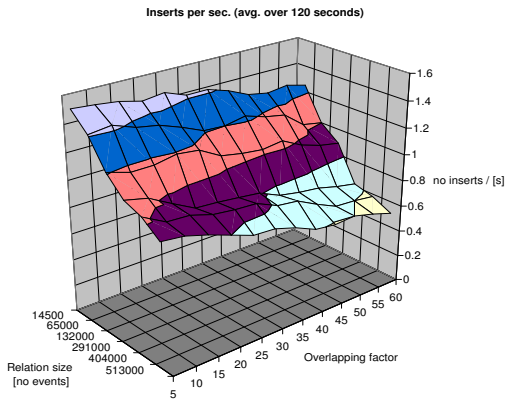

Figure 5. PAM-Insert performance

Though there has been a long history of work on uncertainty in databases [9, 7], the first to recognize the subtle interplay between time and uncertainty were Dyreson and Snodgrass [5] who, in a pioneering paper, laid out a large set of motivating examples and proposed a probabilistic model for temporal data. They extended temporal relational DBMSs to include probabilities about when an event might occur. They proposed an extension of SQL to query such databases and came up with elegant structures to store PDFs. Their work assumed that events were independent. To address this, Dekhtyar et. al. [3] proposed a temporal-probabilistic DB algebra in which they showed how such independence assumptions could be eliminated. The formalisms of both [5, 3] assume that uncertainty occurs only in the temporal attributes. In this paper, our GPT model allows uncertainty to occur both in the temporal attributes, as well as in the data attributes of relations. Our notion of a probabilistic context allows the user to make assumptions about the relationships between events when he asks a query - the GPT data model supports answering queries based on any such probabilistic context.

Our second (and really the primary) contribution focuses on aggregate computations in GPT-databases. Past work on aggregates focused either solely on temporal data [15, 6, 16] or on probabilistic data [11].

We should add that there has been a long history of work on reasoning about both time and uncertainty in the AI community [1, 2, 4, 10] but none of this work addresses aggregate computation.

In short, in this paper, we have proposed a model for aggregate computation in GPT databases that allow us to represent the output of statistical models of supply and demand in a database and then to process all kinds of interesting aggregate queries. Our algorithms have all been implemented and work very efficiently.

References

[1] C. Baral, N. Tran, and L. Tuan. Reasoning about actions in a probabilistic setting. In *Proc. of AAAI'02*, pages 507–512, Edmonton, Alberta, Canada, 2002. AAAI Press.

[2] T. Dean and K. Kanazawa. Probabilistic Temporal Reasoning. In *Proceedings AAAI*, pages 524–529, St. Paul, MN, USA, 1988. AAAI Press / The MIT Press.

[3] A. Dekhtyar, R. Ross, and V. S. Subrahmanian. Probabilistic temporal databases, i: algebra. *ACM Trans. Database Syst.*, 26(1):41–95, 2001.

[4] D. Dubois and H. Prade. Processing Fuzzy Temporal Knowledge. *IEEE Transactions on Systems, Man and Cybernetics*, 19(4):729–744, 1989.

[5] C. E. Dyreson and R. T. Snodgrass. Supporting valid-time indeterminacy. *ACM Trans. Database Syst.*, 23(1):1–57, 1998.

[6] J. A. G. Gendrano, B. C. Huang, J. M. Rodrigue, B. Moon, and R. T. Snodgrass. Parallel algorithms for computing temporal aggregates. In *ICDE*, pages 418–427, 1999.

[7] L. V. S. Lakshmanan, N. Leone, R. Ross, and V. S. Subrahmanian. Probview: a flexible probabilistic database system. *ACM Trans. Database Syst.*, 22(3):419–469, 1997.

[8] L. V. S. Lakshmanan and N. Shiri. A parametric approach to deductive databases with uncertainty. *IEEE Transactions on Knowledge and Data Engineering*, 13(4):554–570, 2001.

[9] L. V. S. Lakshmanan and N. Shiri. A parametric approach to deductive databases with uncertainty. *Knowledge and Data Engineering*, 13(4):554–570, 2001.

[10] D. Lehmann and S. Shelah. Reasoning with time and chance. *Information and Control*, 53:165–198, 1982.

[11] R. Ross, V. S. Subrahmanian, and J. Grant. Aggregate operators in probabilistic databases. *J. ACM*, 52(1):54–101, 2005.

[12] S. M. Ross. *A first course on probability*. Prentice Hall College Div, 1997.

[13] N. Wilkinson. *Managerial Economics: A Problem-Solving Approach*. Cambridge University Press, 2005.

[14] C. D. Wolfram. Strategic bidding in a multi-unit auction: An empirical analysis of bids to supply electricity. *RAND Journal of Economics*, 29(4):703–72, 1998.

[15] J. Yang and J. Widom. Incremental computation and maintenance of temporal aggregates. *The VLDB Journal*, 12(3):262–283, 2003.

[16] D. Zhang, A. Markowetz, V. J. Tsotras, D. Gunopulos, and B. Seeger. Efficient computation of temporal aggregates with range predicates. In *Symposium on Principles of Database Systems*, 2001.

9 Annex I: GPT Algebra

In this section we formally define algebraic operations for GPT relations. The GPT algebra is often directly related to the TP-algebra defined in [3]; however, since GPT relations use a different data structure for tuple data which incorporates uncertainty, the treatment of such data is often different. We will define the GPT algebra in terms of its semi-annotated counterpart for simplicity.

Before defining the GPT algebraic operations, we will remind the reader some of the important concepts used in [3] and how they can be adapted to GPT tuples. A combination function χ is a function that takes a set of probability intervals and returns one probability interval; for example, the extension of a probabilistic conjunction or disjunction strategy to sets of probability intervals is a combination function. Two different tuples in a semi-annotated GPT relation ann_r are considered data identical if the values of all attributes *except* the event Id and the probability interval attributes are identical. Intuitively, two data identical tuples in ann_r refer to the same event, but with different probabilities.

Definition 16 (Compact relations) *A semi-annotated GPT relation ann_r is said to be* compact *if it contains no data identical tuples. A GPT relation R is said to be compact iff $SANN(R)$ is compact.*

Compaction is relevant for multiple reasons. First, a compact relation is preferable to an uncompacted one as storage space is concerned. Second, certain operations such as intersection are interested in tuples that refer to the same event (i.e. data identical tuples). Note that uncompacted GPT relations differ from uncompacted TP-relations. A GPT relation R is uncompacted iff $\exists\ t \neq t' \in R$, $t = (Id, \Gamma, \langle V_1, f_1\rangle, \cdots, \langle V_k, f_k\rangle)$, $t = (Id', \Gamma', \langle V'_1, f'_1\rangle, \cdots, \langle V'_k, f'_k\rangle)$ such that $\Gamma \cap \Gamma' \neq \emptyset$ and $\forall i \in [1, k]$, $V_i \cap V'_i \neq \emptyset$. The justification follows directly from Definition 9. We will now introduce compaction functions[10].

Definition 17 (Semi-annotated compaction) *A function κ from semi-annotated relations to semi-annotated relations is called a compaction function if it satisfies the following axioms:*

- *Compactness: $\kappa(sann_r)$ is compact for all semi-annotated relations $sann_r$.*
- *No fooling around (NFA): If $sann_r$ is compact, then $\kappa(sann_r) = sann_r$.*
- *Conservativness: If $at = (Id, eTime, L_t, U_t, A_1, L_1, U_1, \cdots, A_k, L_k, U_k) \in \kappa(sann_r)$, then $\exists at' = (Id', eTime, L'_t, U'_t, A_1, L'_1, U'_1, \cdots, A_k, L'_k, U'_k) \in sann_r$.*

The semantics of semi-annotated compaction followed those defined for compaction in the TATA algebra in [3]. The semantics of compaction for GPT relations is similar:

[10] First defined in [3] for TP relations.

Definition 18 (GPT compaction) . *A function κ from GPT relations to GPT relations is called a compaction function if it satisfies the following axioms:*

- *Compactness: $\kappa(r)$ is compact for all GPT relations r.*
- *NFA: If r is compact then $SANN(r) = SANN(\kappa(r))$.*
- *Conservativness: If $at = (Id, eTime, L_t, U_t, A_1, L_1, U_1, \cdots, A_k, L_k, U_k) \in \kappa(SANN(r))$, then $\exists at' = (Id', eTime, L'_t, U'_t, A_1, L'_1, U'_1, \cdots, A_k, L'_k, U'_k) \in SANN(r)$.*

The simples compaction functions are based on combination functions. In semi-annotated relations, we require that the probabilities in data identical tuples be combined - hence combination functions are the obvious candidate. [3] gives algorithms for compacting both annotated and TP-relations and their algorithms extend easily to the case of GPT relations. Although in the GPT algebra we can envision using different combination functions for the probability intervals of different attributes, for reasons of simplicity we will denote by κ_χ the compaction function based on the combination function χ[11]. We are now ready to defined the GPT and semi-annotated algebraic operations.

Definition 19 (Semi-annotated intersection) *Let $sann_r$ and $sann'_r$ be two semi-annotated GPT relations with the same schema and let χ be an arbitrary combination function. The intersection $sann''_r =_{def} sann_r \cap_\chi sann'_r$ is a semi-annotated GPT relation such that $\forall t \in \kappa_\chi(sann_r), t' \in \kappa_\chi(sann'_r)$, such that t and t' are data identical, $\exists t'' \in sann''_r$, $t'' = \kappa_\chi(\{t, t'\})$.*

Intuitively, the intersection operation defined above attempts to locate tuples that refer to the same event (i.e. data identical tuples) and compact them into the resulting semi-annotated relation.

Definition 20 (Semi-annotated union) *Let $sann_r$ and $sann'_r$ be two semi-annotated GPT relations with the same schema and let χ be an arbitrary combination function. Let $sann_r \cup sann'_r$ denote the set of tuples in either $sann_r$ or $sann'_r$. The union $sann_r \cup_\chi sann'_r =_{def} \kappa_\chi(sann_r \cup sann'_r)$.*

The definition is intuitive: out of the entire set of tuples in both relations, any two (or more) data identical tuples will be compacted to a single tuple, hence the union semantics.

[11] Which is to be used for all probability intervals.

Definition 21 (Semi-annotated difference) *Let $sann_r$ and $sann'_r$ be two semi-annotated GPT relations with the same schema and let χ be an arbitrary combination function. The difference $sann''_r =_{def} sann_r -_\chi sann'_r$ is a semi-annotated GPT relation such that $\forall t \in \kappa_\chi(sann_r)$ such that $/\exists t' \in \kappa_\chi(sann'_r)$ such that t, t' are data identical, then $t \in sann''_r$.*

Semi-annotated difference produces those tuples in the compaction of the first relation that do not have data identical tuples in the second relation. The semantics are consistent with the previous definitions in that tuples are compared based on the data identical property and not on simple equality.

Definition 22 (Semi-annotated Cartesian product) *Let $sann_r$ and $sann'_r$ be two semi-annotated GPT relations and let α be an arbitrary probabilistic conjunction strategy. The Cartesian product $sann_r \times_\alpha sann'_r$ is a set of tuples at $= (Id'', eTime'', L''_t, U''_t, A_1, L_1, U_1, \cdots, A_m, L_m, U_m, A'_1, L'_1, U'_1, \cdots, A'_n, L'_n, U'_n)$ such that Id'' is a new, dinamically generated event Id and $\forall at \in sann_r \times_\alpha sann'_r$, $\exists t \in sann_r, t' \in sann'_r$ such that:*

(1) $t = (Id, eTime, L_t, U_t, A_1, L_1, U_1, \cdots, A_m, L_m, U_m)$, $t' = (Id', eTime', L'_t, U'_t, A'_1, L'_1, U'_1, \cdots, A'_n, L'_n, U'_n)$.

(2) $eTime'' = eTime = eTime'$.

(3) $[L''_t, U''_t] = [L_t, U_t] \otimes_\alpha [L'_t, U'_t]$.

We will now define selection on semi-annotated relations. For the purpose of this paper, an *atomic selection condition* $\mathcal{C}$ for a semi-annotated relation $sann_r$ can be of the form $(F\ op\ v)$ or $(t_1 \sim t_2)$.

(1) If $F = A_i$ for some data attribute $A_i \in [sann_r]$, $\mathcal{C}$ is a data condition.

(2) If $F = T_j$ for some time unit T_j or if $\mathcal{C}$ is of the form $(t_1 \sim t_2)$, $\mathcal{C}$ is a temporal condition.

(3) If $F =$ "L'' or $F =$ "U'', $\mathcal{C}$ is a probabilistic condition.

A selection condition $\mathcal{F}$ is any combination of atomic conditions using the $\wedge, \vee$ and negation operators.

Definition 23 (Semi-annotated selection) . *Let $sann_r$ be a semi-annotated GPT relation and let $\mathcal{F}$ be a selection condition on $sann_r$. The selection operation $\sigma_\mathcal{F}(sann_r) = \{t \in sann_r | t\ satisfies\ \mathcal{F}\}$.*

Note that unlike other operations defined before, selection does not necessarily produce a compact relation. We will now define the projection operation for semi-annotated relations.

Definition 24 (Semi-annotated projection) *Let $sann_r$ be a semi-annotated GPT relation with schema $[sann_r] = \{Id, eTime, L_t, U_t, A_1, L_1, U_1, \cdots, A_k, L_k, U_k\}$ and let $\mathcal{A} \subseteq \{A_1, \cdots, A_k\}$ be a set of data attributes. Then $\pi_\mathcal{A}(sann_r)$ is a set of tuples such that $\forall t' \in \pi_\mathcal{A}(sann_r)$, $\exists t \in sann_r$ such that:*

(1) $t.Id = t'.Id$, $t.eTime = t'.eTime$, $t.[L_t, U_t] = t'.[L_t.U_t]$.

(2) $\forall A_i \in \mathcal{A}$, $t.A = t'.A$, $t.[L_i, U_i] = t'.[L_i, U_i]$.

(3) There are no other data attributes in $\pi_\mathcal{A}(sann_r)$ except for those in $\mathcal{A}$.

Projection simply filters some of the data attributes in a semi-annotated relation. Note that projection cannot separate a data attribute from its probability interval, nor can it filter out the Id and $eTime$ attributes. We will now introduce the join operation for semi-annotated GPT relations. For simplicity, we only consider the "natural join" operation.

Definition 25 (Semi-annotated join) *Let $sann_r$ and $sann'_r$ be two semi-annotated GPT relations and let $\mathcal{A} = \{A_1, \cdots, A_k\}$ be the set of all data attributes that occur in both $[sann_r]$ and $[sann'_r]$. Let condition $\mathcal{F}$ be a selection condition defined as $(sann_r.A_1 = sann'_r.A_1) \wedge \cdots \wedge (sann_r.A_k = sann'_r.A_k)$. Then the join of $sann_r$ and $sann'_r$ under the α probabilistic conjunction strategy is defined as $\pi_{\mathcal{A}'}(\sigma_\mathcal{F}(sann_r \times_\alpha sann'_r))$, where $\mathcal{A}'$ is the set of all data attributes that occur in either $[sann_r]$ and $[sann'_r]$ after removing duplicate field names.*

At this point, we have defined the algebraic operations for semi-annotated GPT relations. The semantic of GPT algebra operators is based on that of their semi-annotated counterparts.

Definition 26 (GPT unary operations) *Let $op \in \{\sigma_\mathcal{F}, \pi_\mathcal{A}\}$ be a unary operator as defined above and let R be a GPT relation. Then $op(R)$ is a GPT relation such that $\pi_{-Id}(SANN(op(R))) = \pi_{-Id}(op(SANN(R)))$.*

Definition 27 (GPT binary operations) *Let $op \in \{\times_\alpha, -_\chi, \cup_\chi, \cap_\chi\}$ be a binary operator as defined above and let R, R' be two GPT relations such that $R\ op\ R'$ can be performed*[12]*. Then $R\ op\ R'$ is a GPT relation such that $\pi_{-Id}(SANN(R\ op\ R')) = \pi_{-Id}(SANN(R)\ op\ SANN(R'))$.*

[12]For instance, for difference, intersection and union, R and R' need to have the same schema.

The approach taken in defining the GPT algebra based on its semi-annotated version has the advantage of allowing flexibility in providing an efficient implementation of the algebraic operators. For this article, we do not address such algorithms. However, many operators are directly related to the TP-algebra and their implementation is an extension of the one given in [3].