Probability Distributions as Pre-Aggregated Data in Data Warehouses

Igor Timko, Curtis E. Dyreson, and Torben Bach Pedersen

October 19, 2005

TR-14

A DB Technical Report

Title	Probability Distributions as Pre-Aggregated Data in Data Warehouses
	Copyright © 2005 Igor Timko, Curtis E. Dyreson, and Torben Bach Pedersen. All rights reserved.
Author(s)	Igor Timko, Curtis E. Dyreson, and Torben Bach Pedersen
Publication History	October 2005. A DB Technical Report

1 Introduction

Uncertainty is a very important aspect of business data analysis. For example, location-based services (LBSs), e.g., traffic or tourist related services, generate massive amounts of location-based data that must be analyzed in order to optimize and personalize the services. The location-based data is inherently uncertain. For example, in a transportation infrastructure, cars are moving dynamically, so the future location of a car is uncertain. Moreover, the current location is sometimes also uncertain (e.g., known only to a wireless phone cell). For the analysis, of particular interest are *aggregation queries*, e.g., "For each street in Pullman, WA, how many users (in their cars) are in the street?". Current OLAP and data warehouse (DW) technology [14, 16, 27] supports aggregation queries based on a multidimensional data model capturing hierarchies of dimensional data. However, the uncertain data cannot be successfully analyzed with conventional, *deterministic* aggregation queries that do not take the probabilistic nature of this kind of data into consideration. Instead, we need to apply new types of aggregation queries. One such type would compute expected aggregate values (see [26]). However, by computing expected values, a user loses a lot of important information such as extreme cases of aggregate values. Consequently, queries that provide more information must be supported.

The contributions of this chapter are as follows. First, the paper generalizes the notion of *measures* in OLAP data cubes. Specifically, we propose to use *probability distributions* as aggregate values instead of deterministic aggregate values. This approach captures much more information, because a probability distribution can capture a whole range of possibilities of an aggregate value together with their probabilities, while a deterministic aggregate value can only capture either a summary of the whole range (e.g., an expected value) or one characteristic value from the range (e.g., the maximum value). Second, the paper proposes new types of probabilistic OLAP queries that operate on the generalized measure and techniques for processing these queries. Specifically, *aggregation queries* ask for *whole* probability distributions (e.g., "For each street in Pullman, WA, how many cars are in the street?") and *probability queries* ask for summaries about the distributions (e.g., "For each street in Pullman, WA, what is the probability that the number of cars in the street exceeds 50?"). Third, the paper proposes a method for creating probability distributions from fact data. The method enables *pre-aggregation* of our generalized measures. Fourth, the paper proposes a method for using pre-aggregated probability distributions in order to compute higher-level aggregate values based on *convolution*, or "summation", of the probability distributions. Both mentioned methods provide mechanisms for *approximate* computation of the probability distributions, which makes the computation time and space efficient. The methods are implemented in a prototype system and the paper reports on results of initial experiments with the system. The paper thus extends current OLAP/DW technology with means for *probabilistic* data management. The concepts presented in the paper are illustrated using a real-world case study from the LBS domain. The work is based on an on-going collaboration with a leading Danish LBS vendor, Euman A/S [8].

Unlike this paper, which deals with uncertain, or *probabilistic* data, where probabilities are assigned to facts, previous work on *incomplete* data in multidimensional databases [7, 17, 18] focuses on *imprecise* data, where deterministic facts are recorded with different granularities. Deterministic aggregate values are sufficient for analyses of this kind of data. For this reason, unlike this paper, the mentioned papers do not deal with probability distributions as aggregate values. This paper is one of the first to deal with modeling and querying of *probabilistic multidimensional* data. To our best knowledge, apart from our previous paper [26] and this paper, there is only one paper [15] that deals with modeling and querying of this kind of data. However, unlike this paper, the paper [15] deals with deterministic dimensions only and the implementation of the modeling and querying techniques is still in progress.

There is a lot of previous research on *deterministic* aggregation queries (including the mentioned work on incomplete data in multidimensional databases), both for multidimensional and conventional data. However, unlike this paper, much of this work (e.g., the work on spatio-temporal aggregation [19, 29, 30]) does not consider *approximate* query answering, which is very important in dynamic environments where queries have to be answered very quickly or the query result is outdated (e.g., in mobile location-based services). Some papers [11,20,21,25,28] do introduce the approximate query answering into aggregation techniques. However, the *probabilistic* data is still not considered. This paper extends the topics of the mentioned previous research by considering fast, approximate probabilistic aggregation query answering techniques over probabilistic data.

Previous work on probabilistic data management in general [2, 4, 5, 9] handles uncertainty in the data, including joins of relational tables that capture probability distributions. These joins can be used to perform *convolution* of probability distributions. However, the work on joins does not consider fast, *approximate* convolution. This paper does deal with approximate convolution. Outside data management domain, the efficient approximate convolution of probability distributions is considered (e.g., in [22,23]), but this work makes strict assumptions about the distributions such as the number of distributions approaching infinity [22] or uniform samples of their cumulative density functions [23]. This paper considers convolution in more general settings (i.e., convolution of arbitrary finite number of complex cumulative density functions).

The methods presented in this paper are developed for a particular powerful multidimensional model (e.g., a model from paper [26]). The model is an extension of a deterministic multidimensional model from [18].

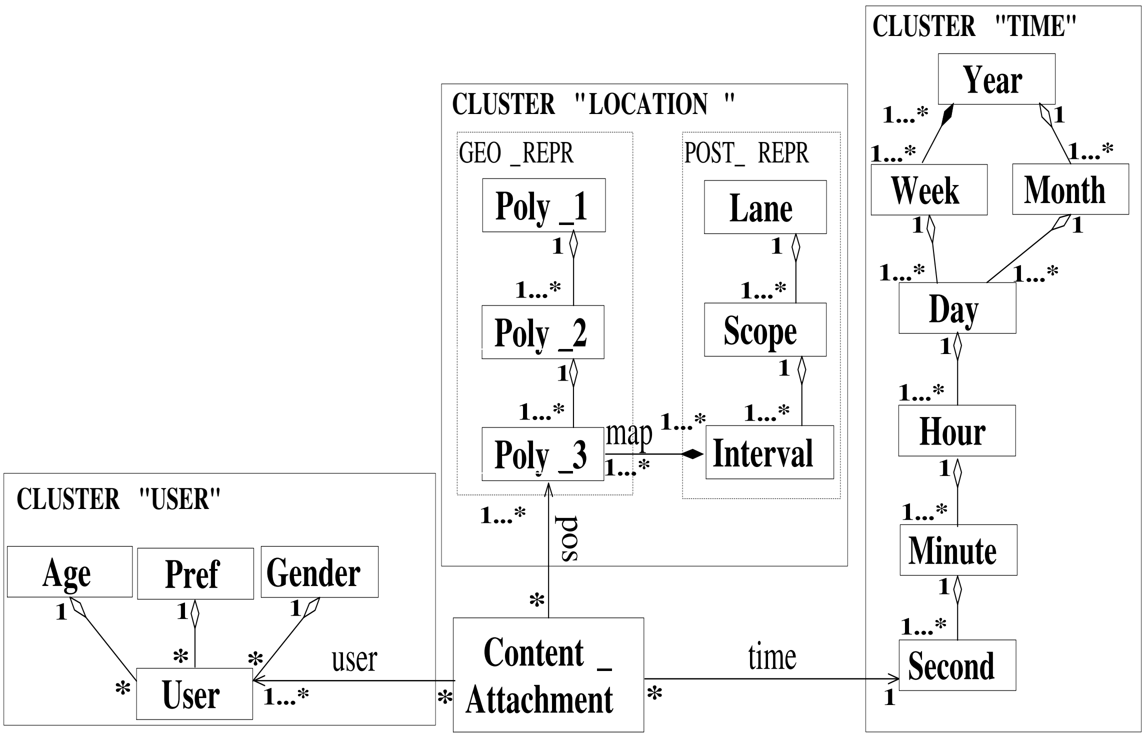


Figure 1: Case Study

The remainder of the paper is structured as follows. Section 2 presents the case study, and describes content and queries. Section 3 briefly introduces the multidimensional data model used as the foundation for the techniques presented in this paper, namely the multidimensional model from paper [26]. Section 4 describes our method for using pre-aggregated probability distributions for further aggregation. Section 5 presents the method for computing the distributions from fact data. Section 6 describes query processing techniques. Section 7 presents result of our experiments with the methods from Sections 4 and 5. Finally, Section 8 concludes the paper and points to future work.

2 Case Study: Location-Based Services

We now discuss the requirements for probabilistic data warehouses by presenting a real-world case study of location-based services (LBS) for which a UML diagram can be seen in Figure 1. First, we discuss the LBS content. Next, we discuss the data that may characterize the content. This data includes content positions within the transportation infrastructure and the time when the positions were reported/recorded. Specifically, we present a transportation infrastructure. We describe two distinct representations of a road network and show how they are related. Content positions are associated with the lowest level in one of the representations. The content and the transportation infrastructure are captured by the data model from paper [26], which we briefly describe in Section 3. This paper focuses on processing of several types of queries, for which the model serves as the foundation. The queries are also discussed in this section.

2.1 Content

We start with discussing content. LBS have both *point* and *interval* content [10]. *Point content*, which is the focus of our paper, concerns entities that are located at a specific geographic location, have no relevant spatial extent, and are attached to specific points in the transportation infrastructure, e.g., traffic accidents, gas stations, and (users' and other's) vehicle positions. *Interval content* concerns data that is considered to relate to a *road section* and is thus attached to intervals of given roads. Examples include speed limits and road surfaces.

Content can be further classified as *dynamic* (frequently evolving) or *static* (rarely evolving). *Static* content, e.g., gas stations or speed limits, remains attached to a point or an interval of a road for a relatively long period of time. In this paper, we focus on very dynamic (*hyper-dynamic*) content, e.g., vehicle positions and their predicted trajectories (which evolve continuously). Positions of static content are usually certain, i.e., *deterministic*, while positions of dynamic content are usually uncertain, i.e., *probabilistic*. For example, if a vehicle position is approximated by a wireless phone cell that covers several roads, then the vehicle may be assumed to be on each of the roads with some *probability*. Furthermore, any position prediction algorithm (for future time queries) will have some degree of uncertainty.

In Figure 1, hyper-dynamic content is modeled by the "USER" cluster, where the "User" class represents users and (implicitly) their vehicles, The "User" class participates in three *full containment relationships* capturing user age, preference, and gender. The users' (vehicle) positions in the infrastructure is modeled by the "LOCATION" cluster. The positions are captured at certain times, represented by the "TIME" cluster. This content positioning/attachment, is modeled as a "Content_Attachment" class which is linked to users, positions, and times. In a probabilistic data warehouse, the "Content_Attachment" class would be a *fact* probabilistically characterized by "USER", "LOCATION", and "TIME" *dimensions*.

2.2 Transportation Infrastructure

We now discuss the aspects of the transportation infrastructure relevant to data aggregation. Different, purpose-dedicated infrastructure representations, may be used, but most modern types of infrastructure representations, e.g., kilometer-post and geographic, are (1) *segment-based* and (2) *hierarchical* [10].

The “LOCATION” cluster from the UML diagram in Figure 1(a) contains two segment-based representations, “GEO_REPR” and “POST_REPR”. The two representations are based on real-world representations used by the LBS company Euman A/S [8]. A detailed discussion of the Euman’s representations can be found in [10]. The “GEO_REPR” and “POST_REPR” are obtained from the corresponding Euman’s representations by representing *lanes* instead of *roads*. Often, lanes of the same road have different characteristics, e.g., different traffic density, so lanes must be captured separately [24]. We refer to segments that capture individual lanes, as *lane segments*. Lane segments may be further subdivided into subsegments to obtain more precise positioning (see Section 2.1). In the “LOCATION” cluster, each such lane segment level is a separate class. “POST_REPR” has three levels: 1) the “Lane” class which captures particular lanes, e.g., a lane on an exit from a highway; 2) the “Scope” class which captures segments between two kilometer posts, which we call *post scopes*, i.e, subdivisions of the road lanes above; 3) the “Interval” class which captures one-meter intervals (of the post scopes above). The “GEO_REPR” also has three levels (but it could have more levels or less levels if needed). Here, a segment is a two-dimensional polyline representing (a part of) a lane. Thus, a segment level is a geographical map. A sequence of segments from the “Poly_3” class (finest scale map), is approximated by (contained in) a segment from the “Poly_2” class (medium scale map), and similarly for “Poly_2” and “Poly_1” (coarsest scale map). The levels define a hierarchy of *full containment (aggregation) relationships* between segments, which is denoted by empty rhombus-headed arrows in our model.

Finally, relationships between the representations must be captured, to allow content attached to one representation to be accessible from another. In the diagram, the relationships between the representations are modeled as ”map” aggregations. Due to differences in how and from what data the representations are built, these mappings are *partial containment relationships*, i.e., segments from the class “Interval” *partially* contain (fully contain is a special case) segments from the “Poly_3” class. A partial containment relationship can also be considered a *probabilistic* relationship. For example, suppose 10% of a segment from the level “Poly_3” is contained in a segment from the level “Interval”. In a probabilistic data warehouse, we may convert this partial containment relationship into the following probabilistic containment relationship: a segment from the level “Poly_3” is contained in a segment from the level “Interval” with the probability of 0.1.

The “position” association captures attachments of user content to level-three segments of “GEO_REPR”, making content mapped to “GEO_REPR” accessible from “POST_REPR”.

2.3 Time

We now discuss the temporal characteristics of content. As mentioned above, content positions are captured at certain *time intervals*, which are organized in a containment hierarchy of temporal granularities, see the “TIME” cluster in Figure 1. Our time hierarchy consists both of *full* and *partial* containment relationships between temporal granularities, e.g., the relationship between hours and days (weeks and years) is full (partial). In a probabilistic data warehouse, these relationships will be captured as probabilistic relationships. User positions are linked to their time intervals by the “time” association.

2.4 Queries

Analytical queries in LBS involve aggregations along multiple hierarchical dimensions, e.g., user content attachments will be aggregated along the USER, LOCATION, and TIME dimensions. As mentioned above, content positions may be given with some uncertainty expressed as probability, and we thus need to evaluate aggregate queries over probabilistic data. Such data cannot be successfully analyzed with conventional, deterministic aggregation queries that do not take the probabilistic nature of the data into consideration. Instead, we need to apply new types of aggregation queries. One such type would compute expected aggregate values (see paper [26]). However, by computing expected values, a user loses a lot of important information such as extreme cases of aggregate values. For this reason, probabilistic data warehouses should support a type of aggregation queries that produce *probability distributions* out of base fact data. For example, the query "For each segment e from the "Poly_2" level, how many cars are on the segment e?" should, for each segment e, compute a probability distribution that would show a whole range of possible numbers of cars on the segment e along with their probabilities. We discuss processing of aggregation queries in Section 6.3.

Next, computing probability distributions from "scratch", i.e., from base fact data, in response to every query could be very expensive for large data warehouses. For this reason, probabilistic data warehouses should support *pre-aggregation* of probability distributions. We discuss creating pre-aggregated data from base fact data in Section 4 and using the pre-aggregated data for answering aggregation queries in Section 5.

Thus, aggregation query results should be probability distributions. However, there are situations where the user may not need to or will not see the whole distribution. For example, the user may only be interested in summary about the distribution. Therefore, a probabilistic data warehouse should support such summary queries on probability distributions, e.g., "For each segment e from the "Poly_2" level, what is the probability that the number of cars on the segment e exceeds 50?". We term such queries *probability queries*. We discuss processing of probability queries in Section 6.2.

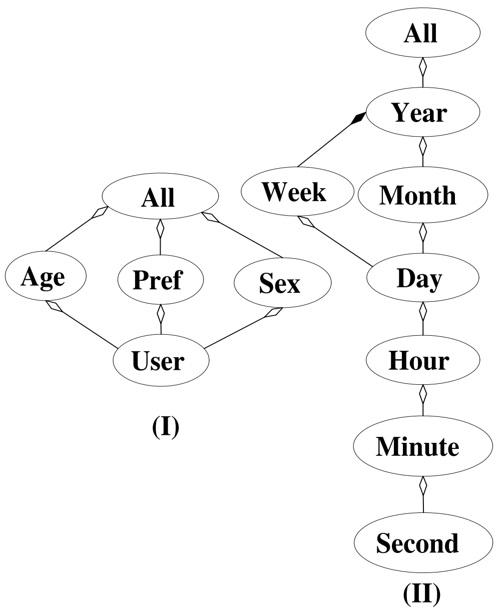
3 Data Model

We now briefly describe the data model from paper [26], which is the foundation for the query processing techniques discussed in this paper. The model has constructs for defining both the *schema* (types) and the *data instances*.

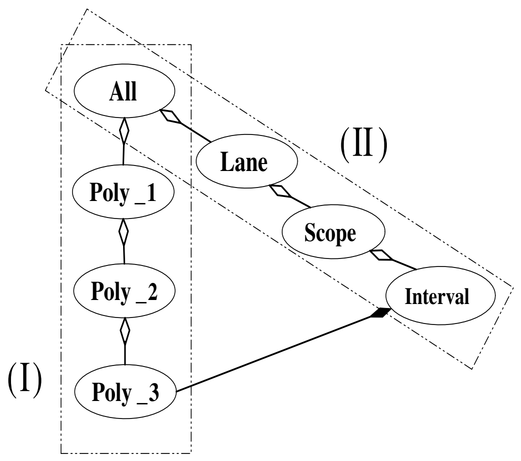
3.1 Data Model Schema

The schema of a cube is defined by a *fact schema* $\mathcal{S}$ that consists of a *fact type* $\mathcal{F}$ (cube name) and a set $\mathcal{D}$ of the *dimension types* $\mathcal{T}_i$ for each dimension.

A dimension type consists of a set $\mathcal{C}_\mathcal{T}$ of the *category types* $\mathcal{C}_j$ (dimension level types), a *relation* $\sqsubset_\mathcal{T}$ on $\mathcal{C}_\mathcal{T}$ specifying the hierarchical organization of the category types, and the special category types $\top_\mathcal{T}$ and $\bot_\mathcal{T}$ that denote the *top* and *bottom* category in the partial order, respectively. For example, a category type $\mathcal{C}$ may be used to model a level of *lane segments*. The partial order on category types, $\sqsubset_\mathcal{T}$, specifies the *partial* (including *full* as a special case) containment relationships among category types. The intuition is to specify whether members of a "child" category type have to be contained in a member of a "parent" category type *fully* or *partially*, e.g., segments from the same (different) representation(s). Next, a *subdimension type* of a dimension type is a set of its category types. Subdimension types of the same dimension type do not intersect except at the $\top_\mathcal{T}$ category type. For example, a subdimension type is used to model a transportation infrastructure representation. The category types from the *same* (*different*) *subdimension type(s)* are related by full (partial) containment relationships.



(a) Dimension types (I) $\mathcal{T}_u$ and (II) $\mathcal{T}_t$



(b) Dimension type $\mathcal{T}_r$ with the subdimension types (I) $\mathcal{T}_g$ and (II) $\mathcal{T}_p$

Figure 2: Dimension types

Example 3.1. Figure 2(a) depicts dimension types $\mathcal{T}_u$ and $\mathcal{T}_t$. In addition, Figure 2(b) depicts a dimension type $\mathcal{T}_r$. The types capture the "USER", "TIME", and "LOCATION" clusters from Figure 1, respectively. Next, the type $\mathcal{T}_r$ has two subdimension types $\mathcal{T}_g$ and $\mathcal{T}_p$, which capture "GEO_REPR" and "POST_REPR", respectively. In Figure 2(b), the "boundary" of each subdimension type is a parallelogram and the types are labeled by (I) and (II), respectively. In the figures, an oval represents a category type. Full (partial) containment category type relationships are given by empty (filled) rhombus-headed arrows. From these *direct* relationships we can deduce the *transitive* relationships between the category types. ▮

3.2 Data Model Instance

In the model *instances*, a *dimension* D consists of a set of *categories*. The *Type* function gives the corresponding type for dimensions and categories. A *subdimension* is an instance of a subdimension type.

A category C_j consists of a set of *dimension values* l_i. The partial order $\sqsubset$ on the union of all values, $\widehat{D}$, specifies the full or partial containment relationships of the values. For example, two values that model segments from the *same* (*different*) representation(s) are usually related by a full (partial) containment relationship. A special value $\top$ in each dimension *fully* contains every other value in the dimension.

Each relationship $l_1 \sqsubset l_2$ has an attached *degree of containment*, $d \in [0;1]$, written $l_1 \sqsubset_d l_2$.

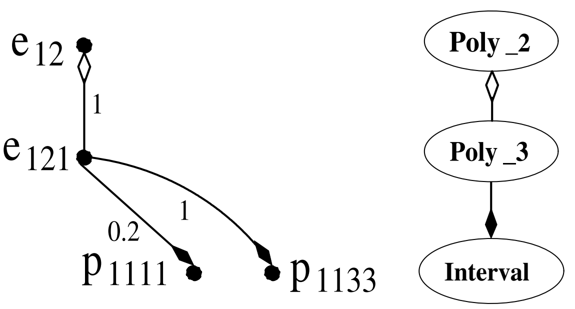


Figure 3: Dimension values of the dimension D_r of type $\mathcal{T}_r$

Example 3.2. Figure 3 depicts a part of dimension D_r of type $\mathcal{T}_r$. In the figure, dimension values are represented by black circles. For each dimension values, its category is represented by an oval that is horizontally aligned with the corresponding black circle. Full (partial) containment relationships between dimension values are given by empty (filled) rhombus-headed arrows. The corresponding degrees of containments are given by numbers by the arrows.

In a given dimension, the degrees have a unique interpretation, but different interpretations are possible. In the following, we present two such interpretations. First, in the following definition, we present a *conservative* interpretation.

Definition 3.3. [**Safe degree of containment**] Given two dimension values l_1 and l_2 and a number $d \in [0; 1]$, the notation $l_1 \sqsubset l_2 \wedge Deg_{saf}(l_1, l_2) = d$ (or $l_1 \sqsubset_d l_2$, for short) means that "l_2 contains at least $d \cdot 100\%$ of l_1". The special case of $d = 0$ means that "l_2 *may* contain l_1, and the size of contained part is unknown". We term d the *safe degree of containment.* ▮

Transitive safe degrees are inferred according to the following principle: we infer the lowest bound on the degree.

The *safe degrees of containment* are useful when the user wishes to infer only those degrees that can be guaranteed, which may be a necessity in some situations. However, in other situations, the user may be interested in the *expected* degrees of containment, which we introduce next. This approach is based on probability theory [6]. We consider each dimension value as a set of points. We deal with the probabilistic events of the form "a value l_1 is contained in a value l_2", which is equivalent to "any point in l_1 is contained in l_2".

Definition 3.4. [**Expected degree of containment**] Given two dimension values l_1 and l_2 and a number $d \in [0; 1]$, the notation $l_1 \sqsubset l_2 \wedge Deg_{exp}(l_1, l_2) = d$ (or $l_1 \sqsubset_d l_2$, for short) means that "l_1 is contained in l_2 with a probability of d". We term d the *expected degree of containment.* ▮

The rule of *transitivity of partial containment with expected degrees* infers the degrees of transitive relationships using probability theory. The basic idea behind the rule is that we obtain the expected degree, expressed as probability, for a transitive relationship by summing up the expected degrees (probabilities) for that relationship obtained through n different aggregation paths.

A *multidimensional object* (cube) consists of a set of *facts* F that are mapped to each dimension, D_j, with a *fact-dimension relation*, $R_j \subseteq F \times D_j$. For a fact $f \in F$ and a dimension value $l \in D_j$, a *fact-dimension relationship*, $(f, l, p_{min}, p_{max}) \in R_j$, means that "*$f$ is inside l* with the probability that belongs to the interval $[p_{min}; p_{max}]$". Next, *fact characterizations*, or inferred fact-dimension relationships, written $f \rightsquigarrow_{[p_{min};p_{max}]} l$, also means that "*$f$ is inside l* with the probability that belongs to the interval $[p_{min}; p_{max}]$". The rules for inferring fact characterizations are based on probability theory. The basic idea behind the rule is that we obtain the probability for a fact characterization by summing up probabilities for that fact characterization obtained through n different aggregation paths.

A Note on Measures Generally, in our data model, dimensions and measures are treated symmetrically, i.e., a dimension from a particular multidimensional object can be used as a domain of an attribute of a multidimensional cell or as a domain of a measure. The model inherits this property from the prototypical model [18]. Specifically, given a measure, its domain is captured in a dimension. For example, in Figure 2(a), the dimension type $\mathcal{T}_u$ contains the AGE category, which can be considered a domain of an AVG aggregation function. Then, in order to associate a fact with a measure value, the fact is attached to the corresponding dimension value from the measure dimension. For example, in order to record a user's age, a fact is attached to a dimension value from the AGE category. The same applies to the types of measures that we consider in this paper, i.e., to probability distributions. However, for simplicity and clarity, in this paper, we hide the "technical" details of capturing probability distributions and treat the distributions as if they reside outside our data model.

4 Using Pre-Aggregated Probability Distributions for Aggregation

In this section, we present a method of using pre-aggregated data for a category for computation of aggregate values for a coarser category. This method is a part of processing aggregate queries extended to handle probability distributions (see Section 6.3). First, we develop the method under the assumption that all of the relationships between dimension values are full, i.e., the relationships are of the form $e_1 \sqsubset_1 e_2$ only. Then, in Section 4.5, we extend the method to support partial containment. In general, the method is based on the well-known notion of *convolution*. The method is general enough to handle pre-aggregated data from probabilistic data warehouses from different domains, but we explain and exemplify the method having the LBS domain in mind.

4.1 Problem Statement

For simplicity and without loss of generality we assume that our data cube has only one dimension. Suppose we are given two categories, C_{from} and C_{to}, such that $C_{from} \sqsubset C_{to}$. Next, suppose we have pre-aggregated data for the category C_{from}. The data is given as follows. For each dimension value $e \in C_{from}$, we have a set, P_e, called the *pre-aggregated count distribution*. The set describes the probability distribution for count for e. More formally,

$$P_e = \{([a_1^e; b_1^e], p_1^e), ([a_2^e; b_2^e], p_2^e), \dots ([a_n^e; b_n^e], p_n^e)\}$$

where $[a_i^e; b_i^e]$ is the i^{th} interval and p_i^e is the probability for the count for e to belong to that interval. The intervals are ordered by their lower bounds in ascending order, i.e., $i < j \Rightarrow a_i \leq a_j$. Furthermore, we allow the intervals to overlap, because we assume an arbitrary methods of producing the distribution. For the same reason, p_i^e may be an arbitrary probability. Since counts are usually whole numbers, we assume that the interval bounds as well as other elements of the intervals are whole numbers. Thus, in this paper, we deal with discrete distributions.

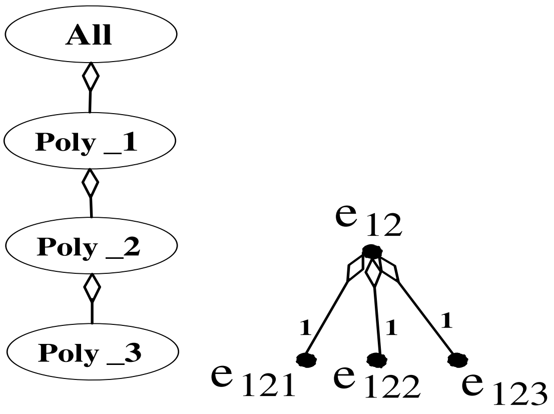


Figure 4: Dimension of a simplified type $\mathcal{T}_r$

Example 4.1. [**Dimension**] In order to exemplify our method more clearly, we simplify the type $\mathcal{T}_r$ from Figure 2(a). The dimension of the simplified type is smaller and has only *full* containment relationships (see Figure 4). Next, suppose $C_{\mathit{from}} = C_{L\_P\_3}$ and $C_{to} = C_{L\_P\_2}$, i.e., given the pre-aggregated data for $C_{L\_P\_3}$, we will aggregate to the level of $C_{L\_P\_2}$. In our case, the pre-aggregated data is given by the three sets $P_{e_{121}}$, $P_{e_{122}}$, and $P_{e_{123}}$. ▮

Before defining aggregate values, we present the definition of *convolution*, which is the basis for our definition of aggregate values.

Definition 4.2. [**Convolution**] Given n random variables, X_1, X_2, ..., and X_n, the *convolution* of X_1, X_2, ..., and X_n is the random variable $Z = X_1 + X_2 + \ldots + X_n$ (also denoted as $Z = \sum_{i=1}^{n} X_i$). ▮

Next, we present the definition of an aggregate value.

Definition 4.3. [**Count distribution**] Given a value $e \in C_{to}$ and a set of *pre-aggregated count distributions*, $P = \{P_{e_1}, P_{e_2}, \ldots, P_{e_m}\}$ such that $e_i \in C_{\mathit{from}} \wedge e_i \sqsubset e$, the *count distribution*, $Count(e)$, is defined as follows:

$$Count(e) = \sum_{i=1}^{m} P_{e_i}$$

According to Definition 4.3, which is a concrete case of Definition 4.2, we compute a probability distribution for a count by convolution of random variables.

Example 4.4. [**Set to convolve**] Continuing Example 4.1, suppose we want to compute the count for e_{12}, $Count(e_{12})$. Then, the set of pre-aggregated count distributions (or the corresponding variables) is $P = \{P_{e_{121}}, P_{e_{122}}, P_{e_{123}}\}$. Thus, $Count(e_{12}) = P_{e_{121}} + P_{e_{122}} + P_{e_{123}}$. ▮

A set P_{e_i} from Definition 4.3 looks similar to a *histogram* [21]. However, we choose to consider a set to be a *probability distribution* of a random variable rather than a histogram. There are several reasons for that. First, a histogram is usually used as a compact representation of a relational table or a *whole* set of OLAP aggregate values in a data cube, while we use the set P_{e_i} to represent a probability distribution of a *single* OLAP aggregate value. Second, we use the convolution operation to perform OLAP aggregation in a convenient way. However, convolution is defined on random variables (or their probability distributions) and not defined on histograms.

4.2 Convolution

In this section, we present a method for efficiently computing the count distribution from Definition 4.3. The method assumes that the distributions from the set P are *independent*. This is a practical assumption, because if the distributions are, in fact, *dependent*, we still compute a useful approximation of the count distribution. A more detailed discussion of that can be found in Section 6.3.

It is extremely inefficient to compute the convolution according to Definition 4.2, i.e., simply adding one variable from the set P at a time. Specifically, if the number of distributions in the set P is m and the number of intervals in each distribution is n, after each addition the number of intervals in the intermediate result will grow by a factor of n, reaching n^m after the convolution is finished. This means that the time needed to compute the convolution is exponential in the number of distributions in the set P.

Thus, we need to reduce the number of intervals to consider. At the same time, we must keep the result correct. We do that by *coalescing* adjacent and intersecting intervals after each addition. Generally, coalescion takes several intervals, returns their union and assigns the sum of their probabilities to the probability of the result. This reduces the precision of the result, but still keeps the result correct. The pseudocode for the computation of the convolution is presented in Algorithm 4.1.

Algorithm 4.1 Count

```
Require: In: $P = \{P_{e_1}, P_{e_2}, \ldots, P_{e_m}\}$
Require: Out: $R = \{([\alpha_1, \beta_1], r_1), ([\alpha_2, \beta_2], r_2), \ldots, ([\alpha_\delta, \beta_\delta], r_\delta)\}$
1: $R \leftarrow P_{e_1}$
2: for all $i \in \{2, \ldots, m\}$ do
3:     $R \leftarrow Add(R, P_{e_i})$
4:     $\delta \leftarrow GetDelta(R)$
5:     if $|R| > \delta$ then
6:         $R \leftarrow Coalesce(R, \delta)$
7: return $R$
```

The algorithm *Count* takes one parameter: a set of pre-aggregated count distributions, P, which is defined earlier. First, the algorithm takes the first and second distributions from P, P_{e_1} and P_{e_2} and computes their convolution (line 3). Second, the algorithm decides on a natural number, δ, which limits the number of elements in the resulting distributions (line 4). (In general, the lower the value of δ, the higher the system performance is. On the other hand, the higher the value of δ, the more informative the coalesced distribution is. Ideally, the system should aim at a balance between the performance and the informativeness of the distribution. We leave details of strategies for deciding on the parameter δ for future work.) Third, the algorithm coalesces the result, R, so that the number of intervals in it is no greater than δ (line 6). Then, the algorithm repeatedly takes the next pre-aggregated count distribution from P, P_{e_i}, and performs the described sequence of actions with the two distributions, P_{e_i} and R.

In line 06, the procedure *Add* adds two distributions. Specifically, suppose we are given two distributions,

$$P_1 = \{([a_1; b_1], p_1), ([a_2; b_2], p_2), \ldots, ([a_n; b_n], p_n)\}$$

and

$$P_2 = \{([x_1; y_1], q_1), ([x_2; y_2], q_2), \ldots, ([x_n; y_n], q_n)\}$$

Then, for each i and j, we add intervals $[a_i; b_i]$ and $[x_j; y_j]$ according to the rules of interval arithmetics [13]. Specifically, for each i and j, $[a_i; b_i] + [x_j; y_j] = [\alpha_{ij}; \beta_{ij}]$, where $\alpha_{ij} = a_i + x_j$

and $\beta_{ij} = b_i + y_j$. The probability $p_i \cdot q_j$ is assigned to the resulting interval. In summary, the result of the procedure is a probability distribution defined as

$$R = \{([\alpha_{ij}; \beta_{ij}], p_i \cdot q_j), i = 1, \dots n, j = 1, \dots, n\}$$

Note that in R the intervals may intersect, because the intersecting intervals still correctly represent the probability distribution for count.

Example 4.5. [**Add**] Continuing Example 4.1, we demonstrate the working of the procedure *Add*. Suppose we have the following pre-aggregated count distributions in the collection P:

1. $P_{e_{121}} = \{([0; 10], 0.2), ([6; 15], 0.3), ([16; 30], 0.5\}$ and
2. $P_{e_{122}} = \{([5; 12], 0.3), ([10; 20], 0.3), ([21; 25], 0.4)\}$.

When we convolve distributions in P by the procedure $Count$, the first ($i = 2$), iteration of the loop in line 04, will produce the following distribution:

$$\begin{aligned} R^{ex} &= Add(P_{e_{121}}, P_{e_{122}}) \\ &= \{([0+5; 10+12], 0.2 \cdot 0.3), ([10; 30], 0.06), ([11; 27], 0.09), \\ &\quad ([16; 35], 0.09), ([21; 35], 0.08), ([21; 42], 0.15), ([26; 50], 0.15), \\ &\quad ([27; 40], 0.12), ([37; 41], 0.2)\} \end{aligned}$$

Note how we obtained the first interval, $[5; 22]$, from R^{ex}: the interval is the sum of the first interval from $P_{e_{121}}$, $[0; 10]$, and the first interval from $P_{e_{122}}$, $[5; 12]$. The interval's lower bound, 5, is the sum of the other two intervals' lower bounds, 0 and 5, and, analogously, the interval's upper bound, 22, is the sum of the other two intervals' upper bounds, 10 and 12. The $[5; 22]$ interval's probability is the product of the other two interval's probabilities. The intuition is as follows: if we know that the number of cars on a segment e_{121} is between 0 and 10 with probability of 0.2 and that the number of cars on a segment e_{122} is between 5 and 12 with a probability of 0.3, then (assuming that a car's location is independent of other cars' locations) we infer that the total number of cars on both segments is between 5 and 22 with a probability of 0.6. ▮

4.3 Coalescion

Algorithm 4.2 Coalesce

```
Require: In: $R = \{([a_1; b_1], p_1), ([a_2; b_2], p_2), \dots, ([a_n; b_n], p_n)\}$, $\delta \in N$
Require: Out: $R_{out}$
 1: $m_\delta \leftarrow GetMaxBestInt(R, \delta)$
 2: $L_\delta \leftarrow GetNumberOfGroups(R, \delta)$
 3: $R_{best} \leftarrow FindBestIntervals(R, m_\delta)$
 4: $G \leftarrow BuildGroups(R, L_\delta)$
 5: $R_r \leftarrow \emptyset$
 6: for all $G_j \in G$ do
 7:     $U_j \leftarrow Unite(G_j \setminus R_{best})$
 8:     $R_r \leftarrow R_r \cup U_j$
 9: $R_{out} \leftarrow R_r \cup R_{best}$
10: return $R_{out}$
```

In line 6 of the procedure $Count$ (see Algorithm 4.1), the procedure $Coalesce$ coalesces intervals from a distribution. The pseudocode of the procedure is presented in Algorithm 4.2. In the following, we describe the algorithm in detail.

Coalescion is performed as follows. Suppose we have to coalesce a distribution:

$$R = \{([a_1; b_1], p_1), ([a_2; b_2], p_2), \dots, ([a_n; b_n], p_n)\}$$

First, in line 3, the procedure $FindBestIntervals$ finds a set of "best" intervals from R, R_{best}, i.e., the intervals that should not be coalesced. In principle, we allow arbitrary definitions of a "good" interval. However, in Example 4.6, we propose a particular definition, HIGH_UNIT_PROB, which is useful from a practical point of view.

Example 4.6. [**HIGH_UNIT_PROB**] Since it is very important to preserve the "shape" of the distribution, e.g., local maximums of its density function, a "good" interval, $([a; b], p)$ should have a *high unit probability*, $p_u = \frac{p}{b-a+1}$. If we assume that all the elements inside the interval have the same probability, the unit probability indicates the probability of a single element of the interval. Thus, the procedure $FindBestIntervals$ returns a set of intervals that have unit probabilities that exceed a certain threshold, p_u^{th}. The threshold, p_u^{th}, is inversely proportional to the average unit probability, i.e., $p_u^{th} = \frac{\rho}{l}$, where $l = \sum_{i=1}^{n} b_i - a_i + 1$ and ρ is a constant. ∎

Since our primary goal is to limit the number of intervals in the resulting distribution, R_{out}, the cardinality of R_{best}, $|R_{best}|$, should not exceed a number that is less than δ, m_δ. This means that we take the top m_δ "best" intervals. The value of m_δ is determined by the procedure $GetMaxBestInt$ in line 1. We leave the details of the procedure for future work.

Second, in line 4, the procedure $BuildGroups$ partitions the set R_c into groups. A group is a series of consecutive intervals that will be coalesced. Two intervals from different groups will never be coalesced. Such grouping reduces overlap between the intervals that are obtained from coalescion. The number of groups, L_δ, is determined by the procedure $GetNumberOfGroups$ in line 2. In principle, we allow arbitrary additional requirements for the groups. However, as in the case of the "good" intervals, in Example 4.7, we propose a specific requirement, EQUI_PROB, which is useful from a practical point of view.

Example 4.7. [**EQUI_PROB**] Since it is very important to preserve the "shape" of the distribution, the sum of interval probabilities for each group should be approximately the same, i.e., approximately equal to $\frac{1}{L_\delta}$. ∎

In order to limit the number of intervals in R_{out}, L_δ should be less than δ. Moreover, for the same reason, it should hold that $L_\delta + m_\delta = \delta$.

Third, the loop in line 6 iterates through a set of the groups and performs actual coalescion. For each group, G_j, the procedure $Unite$ unites all its intervals except for the "good" ones, i.e., the ones from R_{best}. The probability of a union is the sum of the probabilities of the united intervals. By this, each group G_j reduces to a smaller set of intervals, U_j.

As mentioned earlier, coalescion reduces the *precision* of the distribution, i.e., instead of indicating two shorter intervals $[a_i; b_i]$ and $[a_j; b_j]$ for the probable count value, we indicate a longer one $[a_i; b_j]$. However, coalescion keeps the distribution *correct*, i.e., the probability to have a count value from the interval $[a_i; b_j]$ is the same $p_i + p_j$ before and after coalescion.

Finally, the procedure $Coalesce$ used with our strategies HIGH_UNIT_PROB and EQUI_PROB is relatively fast. The procedure's time complexity is $O(n)$, where n is the number of intervals of the input distribution, R. Specifically, let us assume that the procedure $GetMaxBestInt$ runs in constant time (e.g., m_δ is determined before the procedure $Coalesce$ starts). Next, the procedure $GetNumberOfGroups$ runs in constant time, the procedures $FindBestIntervals$ and $BuildGroups$ perform a linear scan of the set of intervals, and the loop in line 6 that unites the intervals is also a linear scan.

	G_1^{ex}						G_2^{ex}		
R^{ex}	[5;22]	[10;30]	[11;27]	[16;35]	[21;35]	***[21;42]***	[26;50]	***[27;40]***	***[37;41]***
p	0.06	0.06	0.09	0.09	0.08	***0.15***	0.15	***0.12***	***0.20***
p_u	0.003	0.003	0.005	0.0045	0.005	***0.007***	0.006	***0.009***	***0.04***

Table 1: The distribution, R^{ex}, with "best" intervals identified and groups built, but before uniting

	U_1^{ex}		U_2^{ex}		
R_{out}^{ex}	[5;35]	[21;42]	[26;50]	[27;40]	[37;41]
p	0.38	0.15	0.15	0.12	0.20

Table 2: The final result of coalescion, the distribution R_{out}^{ex}, obtained by uniting the intervals from R^{ex}

Example 4.8. [**Coalesce**] Continuing Example 4.5, we demonstrate the working of the procedure $Coalesce$. Let us continue the first, i.e., when $i = 2$, iteration of the loop in line 2 of the procedure $Count$. Table 1 depicts the distribution R^{ex} that needs to be coalesced. Specifically, the table present the intervals in the row marked by R^{ex} and their corresponding probabilities and unit probabilities in the rows marked by p and p_u, respectively. Suppose we need to reduce the number of intervals to 5, i.e., $\delta = 5$. Then, suppose we decide to guarantee that the top three "best" intervals remain intact, i.e., $m_\delta = 3$, Furthermore, suppose a "good" interval is a one whose unit probability, p_u, exceeds 0.0055, i.e., $p_u^{th} = 0.006$. Then, the number of groups, $L_\delta = \delta - m_\delta = 2$ and, consequently, the sum of the interval probabilities of each group must be approximately $1/2$. According to this, we form two groups, G_1^{ex} and G_2^{ex} (see Table 1). Also, we identify the set of the three "best" intervals, R_{best}^{ex} ($[26;50]$ is the fourth "best" interval, but it was left out, because we need only three). The "best" intervals are shown in bold italic in Table 1.

Table 2 presents the final result of the coalescion, i.e., the distribution R_{out}^{ex}. In the group G_1^{ex}, the first five intervals were united and the last, "best" interval retained. In the group G_2^{ex}, nothing was united, because there was only one "non-best" interval. The corresponding sets of united intervals are shown under U_1^{ex} and U_2^{ex}, respectively.

Note the importance of retaining "best" intervals. For example, before coalescion, we can see that 20% of the probability mass is concentrated in a very short interval, $[37;41]$, which is a very interesting property of the distribution. Moreover, another 12% of the mass is in the interval $[27;40]$. Taking into consideration that the unit probability of $[27;40]$ is 0.009, we can conclude that the probability to have more than 36 cars is more than $5 * 0.009 + 0.20 = 0.245$. If we had chosen to unite $[37;41]$ and $[27;40]$, the unit probability of the resulting interval, $[27;41]$, would have been almost two times smaller, 0.021. In that case, the lower limit for the probability to have more than 36 cars would be merely $0.021 * 5 = 0.105$. ▮

4.4 Data Selection

The method for using pre-aggregated data described so far may be complemented with *data selection*. In essence, we filter the pre-aggregated count distributions so that only data with the required minimum degree of certainty is used for computing higher-level aggregate values. In other words, we consider only the most likely values for counts and assume that other values are not possible at all.

As before, suppose we are given two categories, C_{from} and C_{to}, such that $C_{from} \sqsubset C_{to}$. Next, suppose we have pre-aggregated data for the category C_{from}. The data is given by the collection of pre-aggregated count distributions, P. For a dimension value $e \in C_{\mathit{from}}$, the collection contains a pre-aggregated count distribution, i.e., a set

$$P_e = \{([a_1^e; b_1^e], p_1^e), ([a_2^e; b_2^e], p_2^e), \dots ([a_n^e; b_n^e], p_n^e)\}$$

where $[a_i^e; b_i^e]$ is the i^{th} interval and p_i^e is the probability for the count for e to belong to that interval.

Data selection works as follows. Given a value $e' \in C_{to}$, we compute a count distribution by considering a set of distributions, $P = \{P_e | e \in C_{\mathit{from}} \wedge e \sqsubset e'\}$. Given a degree of uncertainty, $p_d \in [0; 1]$, for each distribution $P_e \in P$, we filter out those intervals that have a probability less than p_d. More formally, for each P_e, we construct the following new distribution:

$$P'_e = \{([a; b], p) | ([a; b], p) \in P_e \wedge p \geq p_d\}$$

Next, we define a new set of distributions as follows:

$$P' = \{P_e^{n\prime} | e \in C_{\mathit{from}} \wedge e \sqsubset e'\}$$

Then, we finish the computation of a probability distribution of the count for the value e' by performing convolution on P' using the algorithm $Count$ from Algorithm 4.1 with P' as the algorithm's parameter. If at least one interval from any of the distributions in P has been filtered out, the sum of interval probabilities of the result of the convolution of P' is less that 1. In this case, the interval probabilities could be normalized so that their sum equals 1.

Example 4.9. [**Data selection**] In this example, we complement convolution from Examples 4.5 and 4.8 with data selection. If we set the number $p_d = 0.3$, the distributions $P_{e_{121}}$ and $P_{e_{122}}$ reduce to the following distributions, respectively: $P'_{e_{121}} = \{([16; 30], 0.5)\}$ and $P'_{e_{122}} = \{([21; 25], 0.4)\}$. Then, $R^{ex\prime} = \{([37; 41], 0.2)\}$. Obviously, in this case we do not coalesce.

Comparing the results of convolution with and without data selection, i.e., the distributions $R^{ex\prime}$ and R^{ex}_{out} (from Example 4.8), we can say that R^{ex}_{out} is a full, correct result, while $R^{ex\prime}$ is approximate, partial result that nevertheless preserves the most interesting property of the full result, i.e., the local maximum represented by the interval $[37; 41]$. ▮

The method can be easily generalized for (1) a set of degrees of uncertainty, with each degree in the set, p_g, assigned to a particular distribution, P_g, and for (2) degrees of uncertainty given as intervals. In addition, a degree of uncertainty may be a function, e.g., p_g may be the maximum probability from the distribution P_g.

The method gives the user and/or the system a clear way of adjusting the degree of data uncertainty. In addition, the data that the user and/or system is not interested in will be discarded, which improves performance. Since some data may be discarded and, consequently, the pre-aggregate values distorted, the method is best suited for *quickly* obtaining approximate, partial aggregate values that provide a hint on a correct, full aggregate values. A useful OLAP scenario would be as follows. By first setting a high value of p_d, the user will obtain a coarse query result. If the obtained coarse result does not provide enough information to the user, in order to obtain more information, the user may refine the result by setting a lower p_d.

4.5 Handling Partial Containment during Aggregation

The method from Section 4.2 computes aggregate values under the assumption that all the relationships between dimension values are *full* containment relationships. In this section, we extend the settings to also allow *partial* containment relationships with both *safe* and *expected* degrees of containment, and we extend the method accordingly.

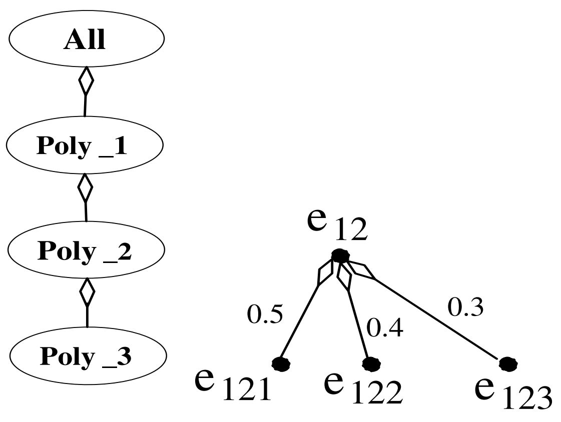


Figure 5: Dimension of a type $\mathcal{T}_r$, with partial containment relationships

In order to exemplify handling partial containment with expected degrees during aggregation, we modify the dimension from previous examples (see Figure 4) to contain partial relationships between dimension values instead of the full relationships. The modified dimension is presented in Figure 5. We implicitly refer to this figure in Examples 4.11 and 4.13.

In the following, we consider cases of expected and safe degrees separately. For both cases, as in Section 4.2, given a value $e \in C_{to}$, we have a collection of pre-aggregated distributions that are used to compute an aggregate value for e. More formally, we have a set $P = \{P_{e_1}, P_{e_2}, \ldots, P_{e_m}\}$ such that $e_i \in C_{\mathit{from}} \wedge e_i \sqsubset_{p_i} e$.

Expected Degrees Let us consider two cases. First, if for any i, $p_i = 1$, i.e., if all the relationships are *full* ones, each pre-aggregate value *surely* contributes to the aggregate value for e. Second, however, if for some i, $p_i < 1$, i.e., if e_i is *partially* contained in e, then the pre-aggregate value P_{e_i} *probabilistically* contributes to the aggregate value for e. Let x_i be an event "the pre-aggregate value P_{e_i} contributes to the aggregate value e". The probability of x_i is p_i. In fact, we also consider the first case as a probabilistic one, where the probability of x_i is 1.

Next, we consider how probabilistic contributions of pre-aggregated values influence the higher level aggregated values. Given an interval $([a; b], p)$ from some P_{e_i} such that the probability of x_i is p_i, we consider an event y defined as "the pre-aggregated count falls into the interval, $[a; b]$, and also contributes to the aggregated value for e". The event y is a conjunction of two independent events: x_a^b (defined as "the pre-aggregated count falls into the interval $[a; b]$") and x_i. Since the probability of x_a^b is p and the probability of x_i is p_i, the probability of y is $p \cdot p_i$. More formally, for pre-aggregated count distribution from the set P,

$$P_{e_i} = \{([a_1^{e_i}; b_1^{e_i}], p_1^{e}), ([a_2^{e_i}; b_2^{e_i}], p_2^{e_i}), \ldots ([a_n^{e_i}; b_n^{e_i}], p_n^{e_i})\}$$

we define a *probabilistic pre-aggregated count distribution*, $P_{e_i}^a$, as follows:

$$P_{e_i}^a = \{([a_1^{e_i}; b_1^{e_i}], p_i \cdot p_1^{e}), ([a_2^{e_i}; b_2^{e_i}], p_i \cdot p_2^{e_i}), \ldots ([a_n^{e_i}; b_n^{e_i}], p_i \cdot p_n^{e_i})\}$$

Intuitively, the probabilistic pre-aggregated count distribution is a distribution of the probability that the count belongs to certain intervals and at the same time contributes to the higher level aggregate value.

Next, we present the definition of an aggregate value.

Definition 4.10. [**Probabilistic count distribution**] Given a value $e \in C_{to}$ and a set of *pre-aggregated count distributions*, $P = \{P_{e_1}, P_{e_2}, \ldots, P_{e_m}\}$ such that $e_i \in C_{\mathit{from}} \wedge e_i \sqsubset_{p_i} e$, the *probabilistic count distribution*, $Count_p(e)$, is defined as follows:

$$Count_p(e) = \sum_{i=1}^{m} P^a_{e_i}$$

where $P^a_{e_i}$ is a probabilistic pre-aggregated count distribution obtained from the pre-aggregated count distribution, P_{e_i}.

Thus, in the case of partial containment relationships between dimension values with expected degrees, an aggregate value is a convolution of the pre-aggregate values that applies a weight to probabilities of the pre-aggregate values. More formally, given a value e, an aggregate value for e is the result of convolution of the following set of distributions: $P^a = \{P^a_{e_1}, P^a_{e_2}, \ldots, P^a_{e_m}\}$. The convolution of this set is performed exactly as described in Section 4.2, i.e., by the algorithm $Count$ from Algorithm 4.1 that takes the set P^a as the input parameter.

Example 4.11. [**Convolution with partial containment with expected degrees**] Analogously to Example 4.4, suppose we want to compute the probabilistic count for e_{12}, $Count_p(e_{12})$. Then, the set of pre-aggregated count distributions (as in the previous examples) is $P = \{P_{e_{121}}, P_{e_{122}}, P_{e_{123}}\}$. Thus, according to Definition 4.10, $Count_p(e_{12}) = P^a_{e_{121}} + P^a_{e_{122}} + P^a_{e_{123}}$. For example, if we have that

$$P_{e_{121}} = \{([0;10], 0.2), ([6;15], 0.3), ([16;30], 0.5\}$$

then the distribution distribution $P^a_{e_{121}}$ is given as follows:

$$\begin{aligned} P^a_{e_{121}} &= \{([0;10], 0.2 \cdot 0.5), ([6;15], 0.3 \cdot 0.5), ([16;30], 0.5 \cdot 0.5\} \\ &= \{([0;10], 0.1), ([6;15], 0.15), ([16;30], 0.25\} \end{aligned}$$

∎

Safe Degrees If for any i, $p_i = 1$, i.e., if all the relationships are *full* ones, each pre-aggregate value *fully* contributes to the aggregate value for e. However, if for some i, $p_i < 1$, i.e., if e_i is only *partially* contained in e, then the pre-aggregate value P_{e_i} also *partially* contributes to the aggregate value for e. The size of the contained part is $p_i \cdot 100\%$ of e_i, so if a full count is c, then it is natural to estimate the contributing part by $p_i \cdot c$. More formally, for any $i = 1, \ldots, m$, we define a *partial pre-aggregated count distribution*, $P^a_{e_i}$, as follows:

$$P^a_{e_i} = p_i \cdot P_{e_i} = \{([p_i \cdot a^{e_i}_1; p_i \cdot b^{e_i}_1], p^e_1), ([p_i \cdot a^{e_i}_2; p_i \cdot b^{e_i}_2], p^{e_i}_2), \ldots ([p_i \cdot a^{e_i}_n; p_i \cdot b^{e_i}_n], p^{e_i}_n)\}$$

Since the interval bounds are integer numbers, the results of multiplications from should include rounding. Next, we present a definition of an aggregate value.

Definition 4.12. [**Partial count distribution**] Given a value $e \in C_{to}$ and a set of *pre-aggregated count distributions*, $P = \{P_{e_1}, P_{e_2}, \ldots, P_{e_m}\}$ such that $e_i \in C_{\mathit{from}} \wedge e_i \sqsubset_{p_i} e$, the *partial count distribution*, $Count_p(e)$, is defined as follows:

$$Count_p(e) = \sum_{i=1}^{m} p_i \cdot P_{e_i} = \sum_{i=1}^{m} P^a_{e_i}$$

Thus, in this case, an aggregate value is a convolution of partial pre-aggregate values. More formally, given a value e, an aggregate value for e is the result of convolution of the following set of distributions: $P^a = \{P^a_{e_1}, P^a_{e_2}, \ldots, P^a_{e_m}\}$. The convolution is performed exactly as described in Section 4.2, i.e., by the algorithm $Count$ from Algorithm 4.1 that takes the set P^a as the input parameter.

Example 4.13. [Convolution with partial containment with safe degrees] Analogously to Example 4.4, suppose we want to compute the partial count for e_{12}, $Count_p(e_{12})$. Then, the set of pre-aggregated count distributions (as in all the previous examples) is $P = \{P_{e_{121}}, P_{e_{122}}, P_{e_{123}}\}$. Thus, according to Definition 4.12, $Count_p(e_{12}) = P^a_{e_{121}} + P^a_{e_{122}} + P^a_{e_{123}}$. For example, if we have that

$$P_{e_{121}} = \{([0;10], 0.2), ([6;15], 0.3), ([16;30], 0.5\}$$

then the distribution distribution $P^a_{e_{121}}$ is given as follows:

$$\begin{aligned} P^a_{e_{121}} &= 0.5 \cdot \{([0;10], 0.2), ([6;15], 0.3), ([16;30], 0.5\} \\ &= \{([0;5], 0.2), ([3;8], 0.3), ([8;15], 0.25\} \end{aligned}$$

▮

5 Computing Pre-Aggregated Probability Distributions

In this section, we present a method for pre-aggregating data. This method is a part of processing aggregate queries extended to handle probability distributions (see Section 6.3). In general, the method is an adaptation of the method for using pre-aggregated data discussed in Section 4. The method is general enough to handle fact data from probabilistic warehouses from different domains. However, we also extend the method to take advantage of LBS domain specifics.

5.1 Basic Method

In the following, we present the problem statement. Suppose we are given a dimension value e. Let $F_e = \{f | f \rightsquigarrow_{[p_{min};p_{max}]} e\}$ be the set of all facts characterized by the value e. So far, we assume that $p^f_{max} = p^f_{min} = p_f$. Our aim is to compute a probability distribution for count for e. Next, we formally define this distribution.

Definition 5.1. [Pre-aggregated count distribution]
Given a dimension value, e, and a set $F_e = \{f | f \rightsquigarrow_{p_f} e\}$, a *pre-aggregated count distribution*, $PreCount(e)$, is defined as follows:

$$PreCount(e) = \{(\alpha, p_\alpha) | \alpha \in N_+\}$$

where

$$p_\alpha = \sum_{F':F' \subseteq F_e \wedge |F'| = \alpha} p_{f^1} \cdot p_{f^2} \cdot \ldots \cdot p_{f^\alpha} \cdot (1 - p_{f^{\alpha+1}}) \cdot (1 - p_{f^{\alpha+2}}) \cdot \ldots \cdot (1 - p_{f^n})$$

and, in turn, f^1, f^2, ..., and f^α belong to F' while $f^{\alpha+1}$, $f^{\alpha+2}$, ..., and f^n belong to $F_e \setminus F'$. ▮

The idea behind Definition 5.1 is as follows. We will denote the probability of an event w by $P(w)$. Let us compute p_α, i.e., the probability that the count for e is equal to α. For a value of the count for e to be equal to α, exactly α facts must be characterized by e and others must be

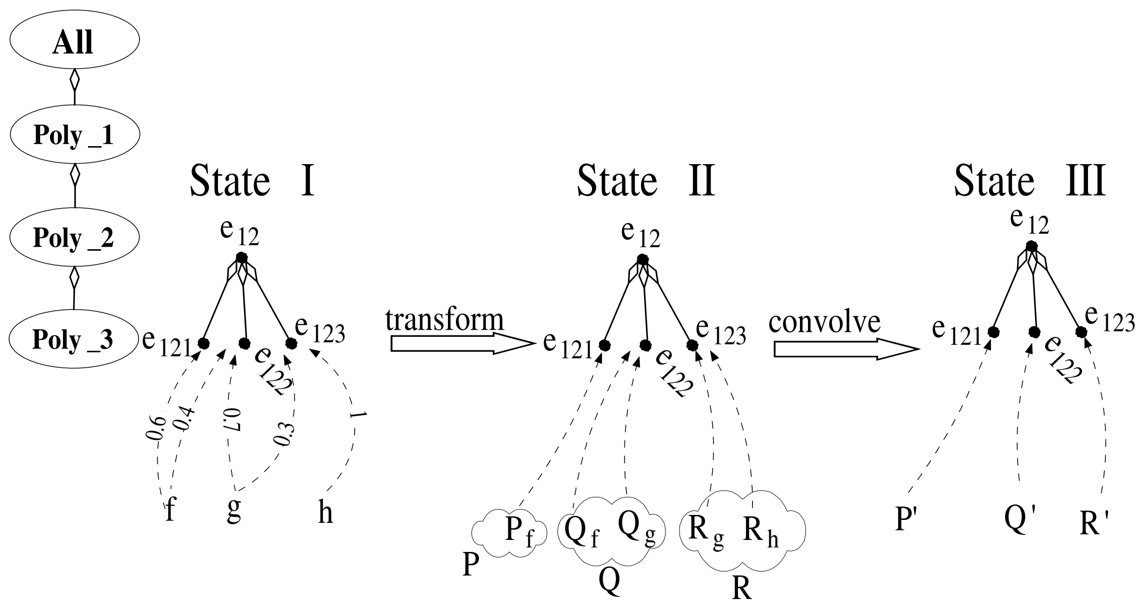


Figure 6: Creating pre-aggregated data

characterized by other values. Given a subset of the set F_e that contains α facts, F', let $w_{F'}$ be the event "all the facts from the set F' are characterized by e and other facts are characterized by other values". For two facts, f_1 and f_2, we assume that their characterizations by a given value, e, are disjoint and independent events. This means that if f^1, f^2, ..., and f^α belong to F' while $f^{\alpha+1}$, $f^{\alpha+2}$, ..., and f^n belong to $F_e \setminus F'$, then $P(w_{F'}) = p_{f^1} \cdot p_{f^2} \cdot \ldots \cdot p_{f^\alpha} \cdot (1 - p_{f^{\alpha+1}}) \cdot (1 - p_{f^{\alpha+2}}) \cdot \ldots \cdot (1 - p_{f^n})$. Now, let v be the event "exactly α facts are characterized by e and other facts are characterized by other values". The event v is the following disjunction of disjoint events: $\bigvee_{F' : F' \subseteq F_e \wedge |F'| = \alpha} w_{F'}$. Therefore,

$$\begin{aligned} p_\alpha &= P(v) \\ &= \sum_{F' : F' \subseteq F_e \wedge |F'| = \alpha} P(w_{F'}) \\ &= \sum_{F' : F' \subseteq F_e \wedge |F'| = \alpha} p_{f^1} \cdot p_{f^2} \cdot \ldots \cdot p_{f^\alpha} \cdot (1 - p_{f^{\alpha+1}}) \cdot (1 - p_{f^{\alpha+2}}) \cdot \ldots \cdot (1 - p_{f^n}) \end{aligned}$$

In the following, we present our method. Computing the count distribution directly according to Definition 5.1 is extremely inefficient. It requires enumerating all subsets of the set F_e. If the number of facts in the set F_e is n, then there are 2^n subsets. This means that time needed for the convolution operation is exponential in the number of facts characterized by e. Such a complexity cannot be handled by large modern real-world warehouses that contain millions or even billions of facts.

In order to compute the count distribution in a more efficient way, we will adapt the method from Section 4. As in Section 4, the efficiency is gained due to coalescion. The adaptation is done in several steps. For a dimension value, e:

1. we consider each fact characterization, $f \rightsquigarrow_{p_f} e$, to be a probability distribution, P_f, that assigns the probability p_f to the interval $[1; 1]$ and the probability $1 - p_f$ to the interval $[0; 0]$. More formally,

$$P_f = \{([0; 0], 1 - p_f), ([1; 1], p_f)\}$$

Given the set of facts, $F_e = \{f | f \rightsquigarrow_{p_f} e\}$, we redefine the set of distributions, P, from Section 4 as the set of distributions P_f, one for each fact from the set F_e. More formally,

$$P = \{P_f | f \in F_e\}$$

2. we convolve the distributions from P by the procedure $Count$ from Algorithm 4.1.

The result of the convolution from Step 2 is the pre-aggregated count distribution for the dimension value e, $PreCount(e)$.

Example 5.2. Figure 6 presents our method for creating pre-aggregated count distributions graphically. The figure depicts three states that the fact data assumes during the process. The states are labeled State I, State II, and State III. The fact data assumes State I before Step 1, State II after Step 1, and Step III after Step 2.

Suppose we need to create pre-aggregate values for the level L_P_3. The facts characterized by dimension values from this level can be seen under State I. The facts form the following sets: $F_{e_{121}} = \{f\}$, $F_{e_{122}} = \{f, g\}$, and $F_{e_{123}} = \{g, h\}$. First, we transform the fact characterizations into the following probability distributions:

$$\begin{aligned}
&f \rightsquigarrow_{0.6} e_{121} \text{ into } P_f = \{([0;0], 0.4), ([1;1], 0.6)\},\\
&f \rightsquigarrow_{0.4} e_{122} \text{ into } Q_f = \{([0;0], 0.6), ([1;1], 0.4)\},\\
&g \rightsquigarrow_{0.7} e_{122} \text{ into } Q_g = \{([0;0], 0.3), ([1;1], 0.7)\},\\
&g \rightsquigarrow_{0.6} e_{123} \text{ into } R_g = \{([0;0], 0.7), ([1;1], 0.3)\},\\
&h \rightsquigarrow_{1} e_{123} \text{ into } R_h = \{([0;0], 0), ([1;1], 1)\}.
\end{aligned}$$

These distributions are grouped into the sets $P = \{P_f\}$, $Q = \{Q_f, Q_g\}$, and $R = \{R_g, R_h\}$, for the values e_{121}, e_{122}, and e_{123}, respectively. These sets are represented by the clouds under State II. Finally, we convolve the distributions from P, Q, and R into the distributions P', Q', and R', respectively, where

$$\begin{aligned}
&P' = \{([0;0], 0.4), ([1;1], 0.6)\},\\
&Q' = \{([0;0], 0.18), ([1;1], 0.54), ([2;2], 0.28)\},\\
&R' = \{([0;0], 0), ([1;1], 0.7), ([2;2], 0.3)\}.
\end{aligned}$$

The result can be seen under State III.
Thus, $PreCount(e_{121}) = P'$, $PreCount(e_{122}) = Q'$, and $PreCount(e_{123}) = R'$. ∎

5.2 Method Extension for Location-Based Services

The basic method from Section 5.1 is quite generic in the sense that it may be applied to fact data from any domain. However, if we consider data from a particular domain, further time-efficiency improvements are possible. In this section, we propose such an improvement for our example *location-based services* domain.

The improvement is based on the assumption that objects in transportation infrastructures, e.g., cars, move in groups, or *units*. In turn, this means that for a given segment, e, a whole unit is either on the segment or outside it. More formally, this means that given a set of facts, $F_e = \{f | f \rightsquigarrow_{p_f} e\}$, we partition the set into units u_1, u_2, ..., u_k. Given a unit $u_i = \{f_1, f_2, \ldots, f_{m_i}\}$ such that $f_1 \rightsquigarrow_{p_1} e, f_2 \rightsquigarrow_{p_2} e, \ldots, f_{m_i} \rightsquigarrow_{p_{m_i}} e$, we define a *unit distribution*, $P_{u_i} = \{(m_i, q_i), (0, 1 - q_i)\}$, where q_i depends on $p_1, p_2, \ldots, p_{m_i}$. Next, by modifying Definition 5.1, we define the *approximate pre-aggregated count distribution*, $PreCount_a(e)$, as

$$PreCount_a(e) = \{(\beta, p_\beta) | \beta \in N_+\}$$

where $p_\beta = \sum_{c_1+c_2+\ldots+c_k=\beta} P_{u_1}(c_1) \cdot P_{u_2}(c_2) \cdot \ldots \cdot P_{u_k}(c_k)$. In turn, $P_{u_i}(c_i)$ denotes q_i and $1-q_i$, if $c_i = m_i$ and $c_i = 0$, respectively.

Given a dimension value, e, the actual computation of $PreCount_a(e)$ may be done as in Section 5.1 with some modifications. Specifically, we redefine the set P as the set of unit distributions, i.e., $P = \{P_{u_1}, P_{u_2}, \ldots, P_{u_k}\}$. Then, we convolve the distributions in the set P.

The described improvement gives a user and/or a system control over time complexity and over precision of the method from Section 5.1. More specifically, by creating groups of size m and thereby reducing the number of distributions in the set P by the factor of m (1) the time complexity (defined as the number of distribution summations) is reduced m times (2) precision is reduced m times, i.e., the result is a set of possible counts $\{m, 2 \cdot m, \ldots, n\}$ instead of $\{1, 2, \ldots n\}$. Note that in this case the precision reduction is a positive feature, because when the number of facts is large, the user will wish to decrease the amount of the received information. Decreasing the precision of the result is a way to do that.

Now we present some ideas on which facts to unite and what probabilities to assign to the units. In general, we believe that it is reasonable to unite those facts that capture positions of objects that exhibit similar *behavior patterns*, e.g., that have the same destination and move at the same speed. Furthermore, all cars that exhibit the same behavior pattern are likely to be close to each other on a road segment, i.e., the cars are on the same subsegment. This means that if pre-aggregation is performed for a value e, then the facts that are characterized by the same child, e', of e could be united. Moreover, similar behavior patterns mean similar probabilities of the corresponding facts. So, the facts with similar probabilities could be united. In summary, the best candidate facts to be united are those, in order of priority, (1) that are marked as having the same behavior pattern, (2) that are characterized by the same child of e, and (3) that are characterized with similar probabilities.

As for probabilities of unit distributions, if *overcounting* is not desirable, then given a unit, the system should choose the *minimum* of probabilities of its elements. More formally, given a unit $u = \{f_1, f_2, \ldots, f_m\}$ such that $f_1 \rightsquigarrow_{p_1} e$, $f_2 \rightsquigarrow_{p_2} e, \ldots, f_m \rightsquigarrow_{p_m} e$, , $P_u = \{(m, q), (0, 1-q)\}$, where $q = min(p_1, p_2, \ldots, p_m)$. Otherwise, the user could choose the *average* of probabilities of the unit's elements. More formally, given a unit $u = \{f_1, f_2, \ldots, f_m\}$, $P_u = \{(m, q), (0, 1-q)\}$, where $q = \frac{p_1+p_2+\ldots+p_m}{m}$. Furthermore, if we know the differences among individual behavior patterns of each fact in the unit, we could assign a weight, w_i, to each probability p_i from the unit.

5.3 Method Generalization

The basic method from Section 5.1 assumed that for any fact generalization, $f \rightsquigarrow_{[p_{min};p_{max}]} e$, we have $p_{min} = p_{max}$. Now we generalize the basic method for arbitrary minimum and maximum probabilities of fact characterizations, i.e., given the set $F_e = \{f | f \rightsquigarrow_{[p^f_{min};p^f_{max}]} e\}$, we assume that $p^f_{max} \geq p^f_{min}$.

We replace the set F_e with two sets, $F^{min}_e = \{f | f \rightsquigarrow_{p^f_{min}} e\}$ and $F^{max}_e = \{f | f \rightsquigarrow_{p^f_{max}} e\}$. Then, we modify Definition 5.1, so that it uses F^{min}_e and F^{max}_e instead of F_e. Thereby, instead of the pre-aggregated count distribution, $PreCount(e)$, we define the *minimum* and *maximum* pre-aggregated count distributions, denoted $PreCount_{min}(e)$ and $PreCount_{max}(e)$, respectively. Then, in order to compute the distributions, we apply the method from Section 5.1 (followed by grouping facts into units, if needed) to the sets F^{min}_e and F^{max}_e, respectively, instead of to the set F_e.

5.4 Data Selection

The method for creating pre-aggregated data may be complemented with *data selection*, analogously to data selection from Section 4.4.

As in Section 5.1, suppose we are given a dimension value e. Let $F_e = \{f | f \leadsto_{[p^f_{min}, p^f_{max}]} e\}$ be the set of all facts characterized by the value e. So far, we assume that $p^f_{max} = p^f_{min} = p_f$.

The data selection is done as follows. Given a *degree of uncertainty*, $p_d \in [0; 1]$, for the set F_e, we filter out those facts that have a probability less than p_d. More formally, we construct a new set of facts, $F'_e = \{f | f \leadsto_{p_f} \wedge p_f > p_d\}$. Then, by replacing F_e with F'_e in Definition 5.1, we define *filtered pre-aggregated count distribution*, $PreCount_f(e)$. In order to compute the filtered pre-aggregated count distribution, we apply the method from Section 5.1 to the set F'_e, instead of to the set F_e.

The method can be easily generalized for (1) a degree of uncertainty given as an interval and (2) probabilities of facts given as intervals, i.e., when $p^f_{max} \geq p^f_{min}$.

The method gives the user and/or the system a clear way of adjusting the degree of data uncertainty. In addition, some data will be discarded, which improves performance. Since some data may be discarded, the method is best suited for the situations where overcounting is not desirable.

6 Query Processing over Probability Distributions

In this section, we discuss how to process queries over probability distributions. Specifically, in Section 6.1, we discuss *preliminary* query processing, i.e., computing required aggregate values. Then, in Section 6.2, we present a new (at least, in the context of OLAP) type of queries, which we call *probability queries*, while in Section 6.3, we consider "standard" OLAP aggregation queries extended to handle probability distributions. Our techniques are general enough to process queries over probabilistic warehouse data from different domain, but we exemplify the techniques with the queries from the LBS domain.

6.1 Preliminary Query Processing

After the system has received an aggregation or a probability query (see Section 2.4) that refers to a group of segments e, e.g., to the segments from "Poly_2" level, it performs preliminary query processing, i.e., computes an aggregate value for each segment, e, from the group. This is done by *integrating the method for using and creating pre-aggregated probability distributions*. Specifically, after the pre-aggregated count distributions have been created by the method from Section 5, they can be used to perform aggregation by the method from Section 4. The reader should refer to the mentioned sections for details, including examples.

Here we note one important point. The pre-aggregated distributions are *dependent*, while the algorithm $Count$ from Algorithm 4.1 assumes that its input is a set of *independent* distributions. This means that the algorithm $Count$ if applied to our pre-aggregate values created by the method from Section 5 returns an approximate result. For instance, continuing Example 5.2, having pre-aggregate values for e_{121}, e_{122}, and e_{123}, i.e., the distributions P', Q', and R', respectively, we compute the aggregate value for e_{12}. If, for clarity, we do not perform full coalescion, but coalesce equal intervals only, then the resulting distribution is

$$S' = \{([1;1], 0.0504), ([2;2], 0.2484), ([3;3], 0.4024), ([4;4], 0.2484), ([5;5], 0.0504)\}$$

The distribution S' is a useful approximation of the correct count, 3, because (1) the probability mass is concentrated around the correct count and (2) the extremely incorrect values, e.g., 1 and 5, have very low probabilities compared to other values. In essence, this is because in the process of convolution, the convolution algorithm rarely encounters pairs of intervals whose sums are extreme intervals.

6.2 Probability Queries

As mentioned in Section 2, the *probability queries* are queries of the form "For each segment e from a level, L, what is the probability, p_x, that the number of cars on the segment e exceeds x?". In this section, we show how to process this type of queries. Specifically, we consider a part of the query, i.e., a query for one particular segment, e.g., "What is the probability, p_x, that the number of cars on a segment e exceeds x?" More formally, let X_e be a random variable whose value is a number of cars on the segment e. Given (1) an aggregate value for a dimension value e, i.e., a distribution

$$P_e = \{([a_1^e; b_1^e], p_1^e), ([a_2^e; b_2^e], p_2^e), \ldots ([a_n^e; b_n^e], p_n^e)\}$$

and (2) a natural number, x, we need to estimate the probability that $X_e > x$.

First of all, in the set P_e, we need to identify the intervals whose probabilities contribute to the result. We will refer to such intervals as *contributing intervals*. Given the query condition $X_e > x$, we denote a set of contributing intervals by P_e^x. We propose two approaches to identifying the contributing intervals. First, according to the *conservative* approach, the *conservative contributing intervals* are those that *fully* lie to the right of x. More formally, the conservative contributing intervals form the following subset of P_e: $P_e^x = \{([a; b], p) : a > x\}$. Second, according to the *liberal* approach, the *liberal contributing intervals* are those that are identified by the conservative approach as well as those that *partially* lie to the right of x. More formally, the liberal contributing intervals form the following subset of P_e: $P_e^x = \{([a; b], p) : b > x\}$. In the following, we will also use the non-conservative, liberal intervals, which form a subset of those liberal intervals that lie partially to the right of x. More formally, the non-conservative, liberal intervals form the following subset of P_e: $P_e^x = \{([a; b], p) : a \leq x \wedge b > x\}$.

After having identified the contributing intervals, we use the probabilities of the contributing intervals to estimate the probability that $X_e > x$. In the following, we propose three approaches to this. According to the first two approaches called *pessimistic* and *optimistic*, the estimate of the probability is the *full* sum of the contributing intervals' probabilities. More formally, if P_e^x is a set of contributing intervals from P_e, then the estimated probability is

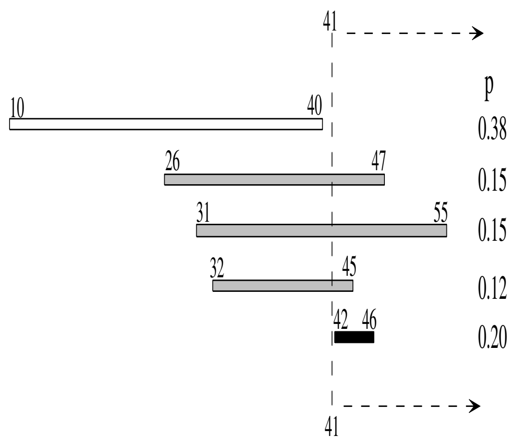
$$p_x = \sum_{(I,p) \in P_e^x} p$$

The difference between the optimistic and the pessimistic approaches is that the former uses *liberal* contributing intervals, while the latter uses *conservative* contributing intervals.

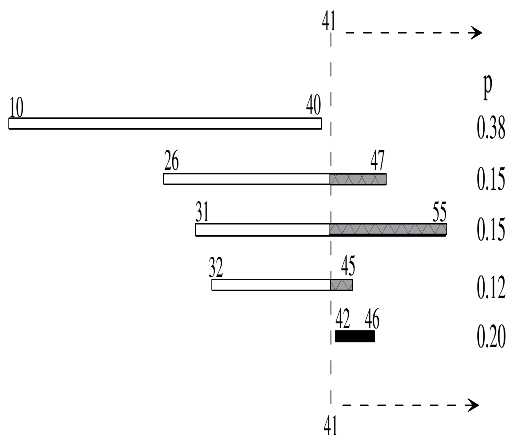
Finally, according to the third approach called *weighted*, the estimate of the probability is the *partial* sum of the contributing interval's probabilities. Specifically, we estimate the probability as follows. First, for a *conservative* contributing interval, its probability fully contributes to the result. Second, for a *non-conservative, liberal* interval, assuming that the interval's probability is uniformly distributed inside the interval, the portion of its probability that lie to the right of x contributes to the result. The sum of all mentioned probabilities is the query result. More formally, if $P_e^{x,c}$ and $P_e^{x,nc}$ is a set of conservative and non-conservative, liberal contributing intervals from P_e, respectively, then the estimated probability is

$$p_x = \sum_{(I,p) \in P_e^{x,c}} p + \sum_{([a;b],p) \in P_e^{x,nc}} \frac{p \cdot (b - x)}{b - a + 1}$$

The above approaches return estimates of the true probability that complement each other. The probability returned by the pessimistic and optimistic approach is a lower and a upper bound of the true probability, respectively. These bounds serve as error bounds and should be shown to the user. In some cases, e.g., when the estimate provided by the error bounds is too uncertain, the user may also want to obtain a better feel of the query result by seeing an approximate value (within the lower and upper bound) around which the true probability is. Such value is returned by the weighed approach.



(a)



(b)

Figure 7: The probability query processing according to (a) pessimistic and optimistic and (b) weighted approach

Example 6.1. [**Probability query processing**] Figure 7 presents our method for probability query processing graphically. In the figure, the intervals from the set P_e are represented by rectangles. The numbers above the rectangles represent interval bounds. Next, the numbers on the side of the rectangles represent interval probabilities. Furthermore, the dashed vertical line represents the number x from the query "What is the probability, p_x, that the number of cars on a segment e exceeds x?" Finally, the dashed horizontal arrows indicate the space where the query should search for contributing intervals.

Figure 7(a) presents the pessimistic and optimistic approach. In the figure, the conservative contributing intervals are black rectangles, i.e., according to the pessimistic approach

$$P_e^x = \{([42; 46], 0.20)\}$$

and the lower bound on the query result is

$$\underline{p_x} = 0.20$$

Next, the liberal contributing intervals are black as well as gray rectangles, i.e., according to the optimistic approach

$$P_e^x = \{([42;46], 0.20), ([32;45], 0.12), ([31;55], 0.15), ([26;47], 0.15)\}$$

and the upper bound on the query result is

$$\overline{p_x} = 0.20 + 0.12 + 0.15 + 0.15 = 0.62$$

Therefore, the first part of the response to the query, which the user receives, is:
"The probability that the number of cars on the segment e exceeds 41 lies in the interval $[0.20; 0.62]$".

Next, Figure 7(b) presents the weighted approach. In the figure, as in Figure 7(a), the conservative contributing intervals are black rectangles, i.e.

$$P_e^{x,c} = \{([42;46], 0.20)\}$$

Next, the non-conservative, liberal rectangles are those rectangles that are partially white and partially gray, shaded, i.e.

$$P_e^{x,nc} = \{([32;45], 0.12), ([31;55], 0.15), ([26;47], 0.15)\}$$

In the gray portions of the rectangles, the probability that contributes to the result is concentrated. Thus, according to the weighted approach

$$p_x = 0.20 + 0.12 \cdot \frac{45-41}{45-32+1} + 0.15 \cdot \frac{14}{25} + 0.15 \cdot \frac{6}{22} = 0.358$$

Therefore, the second part of the response to the query, which the user receives, is: "The probability that the number of cars on the segment e exceeds 41 is approximately 0.358". ▮

The pseudocode for probability query processing according to pessimistic, optimistic, and weighted approach can be seen in Algorithms 6.1, 6.2, and 6.3, respectively. Since the algorithms are very similar, in the following, we discuss them all at once. Each algorithm takes a query condition, x, and a count distribution, P_e, as input parameters and returns query result, i.e., the probability, p_x. The count distribution is sorted in such a way that contributing intervals, if they are present, form a sequence that starts from the first interval. Specifically, for the pessimistic approach, the intervals are sorted by their lower bounds, a_i, in descending order, while for the optimistic and weighted approach, the intervals are sorted by their upper bounds, b_i, in descending order. The general idea of the algorithms is as follows. We go through the sequence of contributing intervals until we reach the end (see the loop in line 3), i.e., until the loop condition fails. For each encountered interval, we add the part of its probability, with which the interval contributes to the result, to the running total, p_x. In the case of the pessimistic and optimistic approaches, this part is always a whole probability, p_i^e (see line 4 in Algorithms 6.1 and 6.2), while in the case of the weighted approach this part is either a whole probability (see line 5 in Algorithm 6.3) or is calculated (see line 7 in Algorithm 6.3).

Queries of the form "For each segment e from a level, L, what is the probability that the number of cars on the segment e does not exceed x?" and "For each segment e from a level, L, what is the probability that the number of cars on a segment e is between x and y?" could be handled analogously.

Algorithm 6.1 Prob_Query_Pessimistic

```
Require: In: $x$, $P_e = \{([a^e_1; b^e_1], p^e_1), ([a^e_2; b^e_2], p^e_2), \dots ([a^e_n; b^e_n], p^e_n)\}$
Require: $i < j \Rightarrow a_i \geq a_j$
Require: Out: $p_x$
1: $p_x \leftarrow 0$
2: $i \leftarrow 1$
3: while $(i \leq n) \wedge (a_i > x)$ do
4:     $p_x \leftarrow p_x + p^e_i$
5:     $i \leftarrow i + 1$
6: return $p_x$
```

Algorithm 6.2 Prob_Query_Optimistic

```
Require: In: $x$, $P_e = \{([a^e_1; b^e_1], p^e_1), ([a^e_2; b^e_2], p^e_2), \dots ([a^e_n; b^e_n], p^e_n)\}$
Require: $i < j \Rightarrow b_i \geq b_j$
Require: Out: $p_x$
1: $p_x \leftarrow 0$
2: $i \leftarrow 1$
3: while $(i \leq n) \wedge (b_i > x)$ do
4:     $p_x \leftarrow p_x + p^e_i$
5:     $i \leftarrow i + 1$
6: return $p_x$
```

Algorithm 6.3 Prob_Query_Weighted

```
Require: In: $x$, $P_e = \{([a^e_1; b^e_1], p^e_1), ([a^e_2; b^e_2], p^e_2), \dots ([a^e_n; b^e_n], p^e_n)\}$
Require: $i < j \Rightarrow b_i \geq b_j$
Require: Out: $p_x$
1: $p_x \leftarrow 0$
2: $i \leftarrow 1$
3: while $(i \leq n) \wedge (b_i > x)$ do
4:     if $a_i > x$ then
5:         $p_x \leftarrow p_x + p^e_i$
6:     else
7:         $p_x \leftarrow p_x + \frac{p^e_i \cdot (b^e_i - x)}{b^e_i - a^e_i + 1}$
8:     $i \leftarrow i + 1$
9: return $p_x$
```

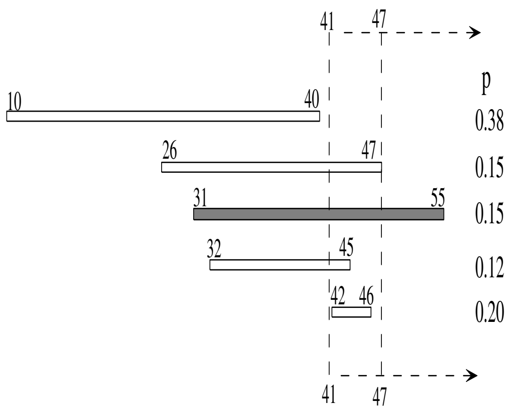


Figure 8: Alternative query processing

Alternative Query Processing Now we discuss an idea of *alternative query processing*. Note that the user may not be satisfied with the result of processing the *original query*, because the estimate provided by the error bounds is not certain enough. In Example 6.1, the processing of the original query, "What is the probability that the number of cars on a segment e exceeds 41?", yields the upper and lower bound of 0.62 and 0.20, respectively. The user may conclude that the difference between the bounds of 0.42 makes the query result too uncertain. In this case, the system could propose an *alternative query* that could be processed with greater certainty (see Figure 8). An example alternative query could be "What is the probability that the number of cars on a segment e exceeds 47?". The alternative query condition was chosen to lower the number of intervals that *partially* lie to the right of the condition. With the original query condition we had three such intervals, while with the alternative query condition we have only one such interval. The response to this query is "The probability that the number of cars on the segment e exceeds 41 lies in the interval $[0; 0.15]$". Now the difference between the error bounds is only 0.15, which may make the result of the alternative query more attractive to the user. We leave the details of the method of *alternative queries* for future work.

6.3 Aggregation Queries

As mentioned in Section 2, *aggregation queries* are the queries of the form "For each segment e from a level, L, how many cars are on the segment e?". In this section, we show how to process aggregation queries. Specifically, we consider a part of the query, i.e., a query for one particular segment, e.g., "How many cars are on a segment e?". A query result is the probability distributions of the count of facts attached to segment e. In general, we reduce an aggregation query, which asks for a whole *cumulative density function* (CDF), to a series of probability queries, so that each probability query "constructs" a piece of the whole function.

Approximation of the Cumulative Density Function As the response to an aggregate query, "How many cars are on segment e?", a user obtains an *approximation of the CDF*, a function $\mathcal{F}_e : [0; \infty) \rightarrowtail [0; 1]$. The approximation is constructed as follows. Given an aggregate value for a dimension value e, i.e., a distribution

$$P_e = \{([a^e_1; b^e_1], p^e_1), ([a^e_2; b^e_2], p^e_2), \ldots ([a^e_n; b^e_n], p^e_n)\}$$

we formulate a series of probability queries, $Q = \{Q_1, Q_2, \ldots, Q_n\}$. Each query, $Q_i \in Q$, is formulated as follows: "What is the probability, p_i, that the number of cars on a segment e is lower than x_i?". (A naive way to process this *series* of queries is to process each query $Q_i \in Q$ separately.

Later in the section, we present an optimized approach that processes the queries in parallel, see "Query Processing".) Then, the following is a set of *sampled points* from the approximation $\mathcal{F}_e$:

$$F_e = \{(x_1, p_1), (x_2, p_2), \ldots, (x_m, p_m)\}$$

where $m \leq n$. We explain how to select the values of x_i shortly.

The set of probabilities, $p_1, p_2, \ldots p_m$, can be defined according to the *pessimistic*, *optimistic*, or *weighted* approach (see Section 6.2), which produces three different sets of sampled points, $\underline{F_e}$, $\overline{F_e}$, and $\widetilde{F_e}$. Out of $\underline{F_e}$ and $\overline{F_e}$, we construct approximations of the true CDF, $\underline{\mathcal{F}_e}$ and $\overline{\mathcal{F}_e}$, that are *lower* and *upper* bound on the true CDF, respectively, while out of $\widetilde{F_e}$, we construct an *average approximation* of the true CDF, $\widetilde{\mathcal{F}_e}$, that lies between the lower bound and the upper bound. Specifically, given the set $F_e = \{(x_1, p_1), (x_2, p_2), \ldots, (x_m, p_m)\}$, we construct the following approximations of the true CDF, $\mathcal{F}_e$.

1. If $p_1, p_2, \ldots p_m$ are defined according to *pessimistic approach*, we construct the *lower bound*, $\underline{\mathcal{F}_e}$, as follows:

$$\underline{\mathcal{F}_e}(x) = \begin{cases} 0 & \text{if } x < x_1 \\ p_i & \text{if } x_i \leq x < x_{i+1}, \text{for } i \in \{1, 2, \ldots, m-1\} \\ 1 & \text{if } x \geq x_m \end{cases}$$

   The idea behind the construction is as follows. First, since the points from F_e are sampled points from some lower bound of the CDF, our lower bound includes these points. Second, our CDF (as any CDF) is a non-decreasing function. Between the two points, x_i and x_{i+1}, we do not know how the values of our CDF grow, but we know that the values cannot be lower than p_i. For this reason, we obtain our lower bound by setting all its values between the points x_i and x_{i+1} to p_i. Third, the minimum value of our CDF (as of any CDF) is 0. Since the values of our CDF before the point x_1 are not known, our lower bound for these values is set to this minimum, 0. Finally, the maximum value of our CDF (as of any CDF) is 1. Since the values of our CDF after the point x_m are known to be 1, our lower bound for these values is set to 1.

2. If $p_1, p_2, \ldots p_m$ are defined according to *optimistic approach*, we construct the *upper bound*, $\overline{\mathcal{F}_e}$, as follows:

$$\overline{\mathcal{F}_e}(x) = \begin{cases} p_1 & \text{if } x < x_1 \\ p_{i+1} & \text{if } x_i \leq x < x_{i+1}, \text{for } i \in \{1, 2, \ldots, m-1\} \\ 1 & \text{if } x \geq x_m \end{cases}$$

   The idea behind the construction is as follows. First, since the points from F_e are sampled points from some upper bound of the CDF, our upper bound includes these points. Second, our CDF (as any CDF) is a non-decreasing function. Between the two points, x_i and x_{i+1}, we do not know how the values of our CDF grow, but we know that the values cannot be higher than p_{i+1}. For this reason, we obtain our upper bound by setting all its values between the points x_i and x_{i+1} to p_{i+1}. Third, since the exact values of our CDF before the point x_1 are not known, but are known to be not higher than p_1, our upper bound for these values is set to p_1. Finally, the maximum value of our CDF (as of any CDF) is 1. Since the values of our CDF after the point x_m are known to be 1, our upper bound for these values is set to 1.

3. If $p_1, p_2, \ldots p_m$ are defined according to *weighted approach*, we construct the *average approximation*, $\widetilde{\mathcal{F}_e}$, as follows:

$$\widetilde{\mathcal{F}_e}(x) = \begin{cases} \frac{x \cdot p_1}{x_1} & \text{if } x < x_1 \\ \frac{(x - x_i) \cdot (p_{i+1} - p_i)}{x_{i+1} - x_i} + p_i & \text{if } x_i \leq x < x_{i+1}, \text{for } i \in \{1, 2, \ldots, m-1\} \\ 1 & \text{if } x \geq x_m \end{cases}$$

The idea behind the construction is as follows. First, since the points from F_e are sampled points from some approximation of the CDF that lies between our upper and our lower bound, our average approximation includes these points. Second, our CDF (as any CDF) is a non-decreasing function. Between the two points, x_i and x_{i+1}, we do not know how the values of our CDF grow, but we know that the values grow approximately between p_i and p_{i+1}. For this reason, we obtain our average approximation by linear interpolation, i.e., by "placing" all its values between the points xi and x_{i+1} on the line that crosses the points. Third, since the exact values of our CDF between the points 0 and x_1 are not known, we again perform linear interpolation, i.e., assume that the values are on the line that crosses these two points. Finally, the maximum value of our CDF (as of any CDF) is 1. Since the values of our CDF after the point x_m are known to be 1, our average approximation for these values is set to 1. So constructed average approximation, $\widetilde{\mathcal{F}_e}$, approximates each value of the CDF at each point, x, by the value between our lower and upper bound, $\underline{\mathcal{F}_e}(x)$ and $\overline{\mathcal{F}_e}(x)$.

As for the values in the set of probability query conditions, $X = \{x_1, x_2, \ldots, x_n\}$, for each $x_i \in X$, we set $x_i = b_i + 1$. The reason for this selection of the query conditions is as follows. For a query, $Q_i \in Q$, when its probability, p_i, is defined by *pessimistic approach*, we would like to find at least one contributing interval. This is why x_i is greater than b_i. Moreover, in an approximation of the CDF, $\mathcal{F}_e$, we would like to capture the "shape" of the CDF as given by the upper bounds of the intervals from P_e. This is why x_i is as close to b_i as possible.

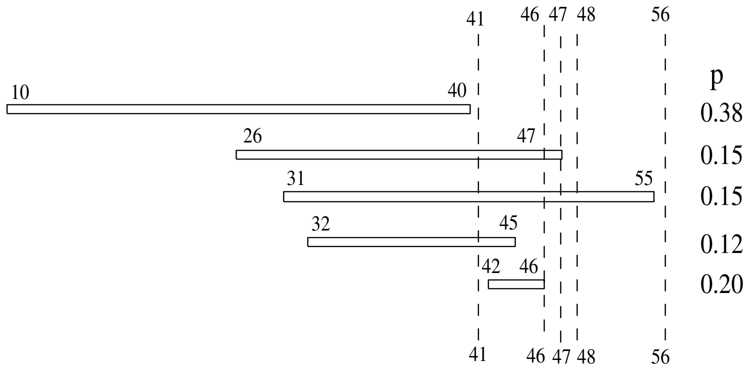


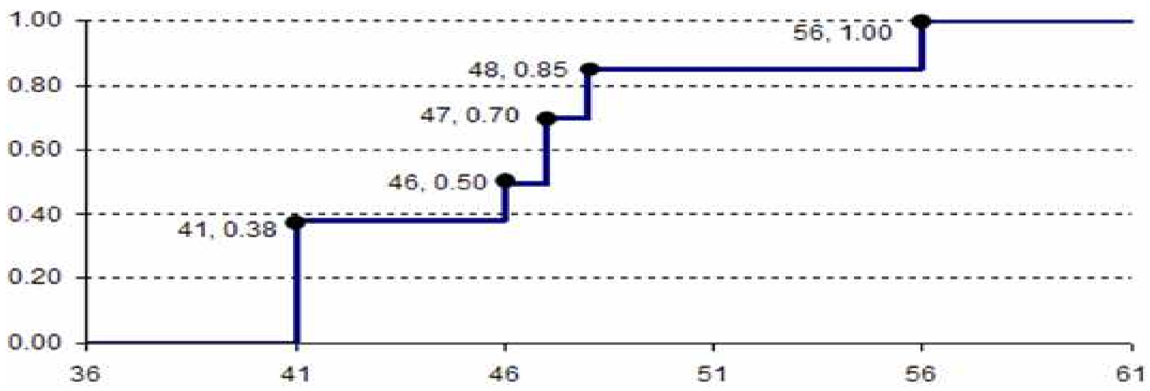
Figure 9: An example count distribution, P_e^{ex}, and the corresponding set of query conditions, X^{ex}

Example 6.2. [**Aggregation query**] Figure 9 presents our method for processing aggregation queries graphically. Suppose we have an example count distribution, P_e^{ex} for a dimension value, e. In the figure, the intervals from this distribution are represented by rectangles. The numbers above the rectangles represent interval bounds. Next, the numbers on the side of the rectangles represent interval probabilities. Furthermore, the dashed vertical lines represent the query conditions $x_i \in X^{ex}$ for the example count distribution from the queries "What is the probability, p_i, that the number of cars on a segment e exceeds x_i?", respectively. If the queries are processed according to the (1) pessimistic, (2) optimistic, and (3) weighted approach, we construct the following sets of sampled points from approximations of the CDF for count for e, respectively:

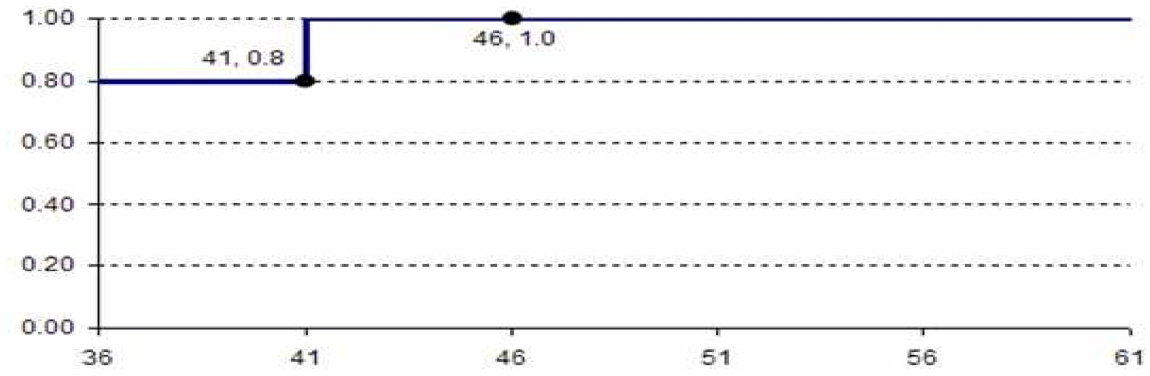
1. $\underline{F_e^{ex}} = \{(41, 0.38), (46, 0.5), (47, 0.7), (48, 0.85), (56, 1)\}$,

2. $\overline{F_e^{ex}} = \{(41, 0.8), (46, 1)\}$, and

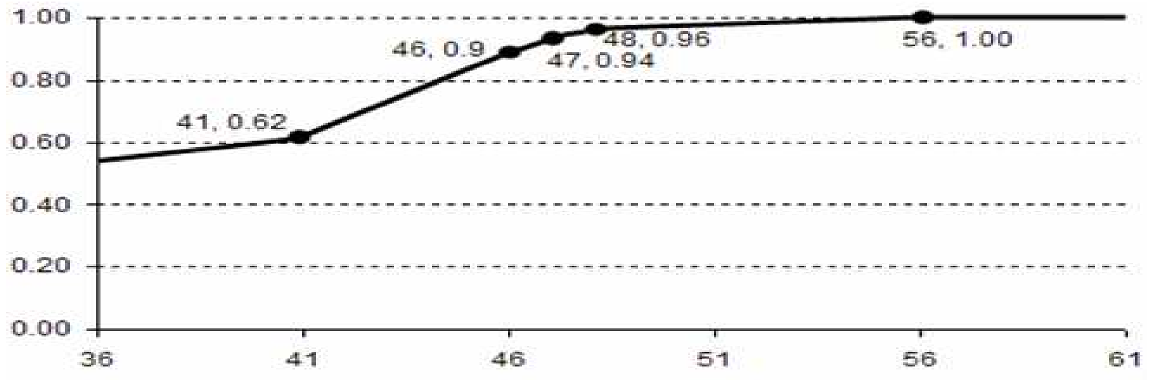
3. $\widetilde{F_e^{ex}} = \{(41, 0.62), (46, 0.9), (47, 0.94), (48, 0.96), (56, 1)\}$.



(a)



(b)



(c)

Figure 10: An example (a) *lower bound approximation*, $\underline{\mathfrak{F}_e^{ex}}$, (b) *upper bound approximation*, $\overline{\mathfrak{F}_e^{ex}}$, (c) *average approximation*, $\widetilde{\mathfrak{F}_e^{ex}}$

Out of these sets of sampled points, we construct the following approximations of the CDF, respectively: (1) the *lower bound*, $\underline{\mathfrak{F}_e^{ex}}$, (2) the *upper bound*, $\overline{\mathfrak{F}_e^{ex}}$, and (3) the *average approximation*, $\widetilde{\mathfrak{F}_e^{ex}}$. The constructed approximations are presented graphically to the user. Figures 10(a), 10(b), and 10(c) present the approximations of the CDF, $\underline{\mathfrak{F}_e^{ex}}$, $\overline{\mathfrak{F}_e^{ex}}$, and $\widetilde{\mathfrak{F}_e^{ex}}$, respectively, as a user would see them. The figures depict the fragments of the approximations where the sampled points are concentrated.

Algorithm 6.4 Aggregation_Query_Pessimistic

```
Require: In: $P_e = \{([a_1^e; b_1^e], p_1^e), ([a_2^e; b_2^e], p_2^e), \ldots ([a_n^e; b_n^e], p_n^e)\}$,
              where $i < j \Rightarrow b_i \leq b_j$
Require: Out: $F_e = \{(x_1, p_1), (x_2, p_2), \ldots, (x_m, p_m)\}$
 1: $F_e \leftarrow \emptyset$
 2: $X \leftarrow \{b_1^e + 1, b_2^e + 1, \ldots, b_n^e + 1\}$
 3: $p_x \leftarrow 0$
 4: $i \leftarrow 1$
 5: for all $x \in X$ do
 6:     while $(i \leq n) \wedge (b_i < x)$ do
 7:         $p_x \leftarrow p_x + p_i^e$
 8:         $i \leftarrow i + 1$
 9:     $F_e \leftarrow (x, p_x)$
10:     if $p_x = 1$ then return $F_e$
```

Algorithm 6.5 Aggregation_Query_Optimistic

```
Require: In: $P_e = \{([a_1^e; b_1^e], p_1^e), ([a_2^e; b_2^e], p_2^e), \ldots ([a_n^e; b_n^e], p_n^e)\}$,
              where $i < j \Rightarrow a_i \leq a_j$
Require: Out: $F_e = \{(x_1, p_1), (x_2, p_2), \ldots, (x_m, p_m)\}$
 1: $F_e \leftarrow \emptyset$
 2: $X \leftarrow SortAscending(\{b_1^e + 1, b_2^e + 1, \ldots, b_n^e + 1\})$
 3: $p_x \leftarrow 0$
 4: $i \leftarrow 1$
 5: for all $x \in X$ do
 6:     while $(i \leq n) \wedge (a_i < x)$ do
 7:         $p_x \leftarrow p_x + p_i^e$
 8:         $i \leftarrow i + 1$
 9:     $F_e \leftarrow (x, p_x)$
10:     if $p_x = 1$ then return $F_e$
```

Algorithm 6.6 Aggregation_Query_Weighted

```
Require: In: $P_e = \{([a_1^e; b_1^e], p_1^e), ([a_2^e; b_2^e], p_2^e), \ldots ([a_n^e; b_n^e], p_n^e)\}$,
              where $i < j \Rightarrow a_i \leq a_j$
Require: Out: $F_e = \{(x_1, p_1), (x_2, p_2), \ldots, (x_m, p_m)\}$
 1: $F_e \leftarrow \emptyset$
 2: $X \leftarrow SortAscending(\{b_1^e + 1, b_2^e + 1, \ldots, b_n^e + 1\})$
 3: $p_c \leftarrow 0$
 4: $R \leftarrow P_e$
 5: for all $x \in X$ do
 6:     $p_x \leftarrow p_c$
 7:     for all $i$ such that $([a_i^e; b_i^e], p_i^e) \in R \wedge a_i \leq x$ do
 8:         if $b_i \leq x$ then
 9:             $p_x \leftarrow p_x + p_i^e$
10:             $R \leftarrow R \setminus ([a_i^e; b_i^e], p_i^e)$
11:             $p_c \leftarrow p_c + p_i^e$
12:         else
13:             $p_x \leftarrow p_x + \frac{p_i^e \cdot (b_i^e - x)}{b_i^e - a_i^e + 1}$
14:     $F_e \leftarrow (x, p_x)$
15:     if $p_x = 1$ then return $F_e$
```

Query Processing Conceptually, each probability query, Q_i, from a series Q (see "Approximation of the Cumulative Density Function") could be processed in isolation. However, it is possible to optimize the processing for a *series* of queries. In general, the optimization is achieved by using the result of the previous query, Q_i, as a partial result of the next query, Q_{i+1}. The pseudocode for the optimized processing is presented in Algorithms 6.4, 6.5, and 6.6 that process the queries according to the *pessimistic*, *optimistic*, and *weighted* approach, respectively. The following discussion considers all the three algorithms at once. Each algorithm takes a count distribution, P_e, as an input and returns a set of sampled points from an approximation of the CDF, F_e. The count distribution is sorted in such a way that contributing intervals, if they are present, for each query condition, $x \in X$, form a sequence that starts from the first interval of the count distribution. Specifically, for the pessimistic approach, the intervals are sorted by their upper bounds, b_i, in ascending order, while for the optimistic and weighted approach, the intervals are sorted by their lower bounds, a_i, in ascending order. In each algorithm, in line 2, the set of query conditions is formed. For the optimized processing, the query conditions should be sorted in ascending order. For this reason, in Algorithms 6.5 and 6.6, we apply the procedure $SortAscending$ for sorting.

After line 2, Algorithms 6.5 and 6.4 differ significantly from Algorithms 6.6. For this reason, in the following, we discuss the two first algorithms and the third algorithm separately. First, we consider Algorithm 6.4 and Algorithm 6.5 The general idea of the algorithms is as follows. For each query condition, $x_i \in X$, if an interval, I, is a contributing interval for x_i, then I is *also* a contributing interval for the next query condition, x_{i+1}. Moreover, according to both the pessimistic and optimistic approach, if an interval, I, is a contributing interval for two or more query conditions, then I contributes *equally*, i.e., with the probability of I, to the result of each probability query given by those query conditions. This means that we need to consider each interval from P_e only once. For this reason, in the algorithm, we maintain a single probability running total , p_x (see line 3), and a single interval counter, i (see line 4). For each query condition, $x \in X$, there is one iteration of the loop in line 5. When $(k-1)$ iterations have finished, we may have considered some contributing intervals of the k^{th} query condition and p_x is the sum of probabilities of these intervals. Suppose i' is the value of the counter i, when $(k-1)$ iterations have finished. During the k^{th} iteration, we go through the remaining sequence of the contributing intervals of the k^{th} query condition (see the loop in line 6), which starts at the interval indicated by i', until we reach the end of the sequence, i.e., until the loop condition fails. For each encountered interval, we add the probability of the interval to the running total, p_x (see line 7). When we reach the end of the sequence, we insert the pair of a query condition and its probability, (x, p_x), into the set of sampled points of the approximation of the CDF, F_e (see line 9). The algorithm terminates when the running probability total, p_x, reaches 1 (see line 10), which happens when all the query conditions have been considered or when all the intervals from P_e are contributing intervals for the remaining, unconsidered query conditions.

In the following, we consider Algorithm 6.6. The general idea of the algorithm is as follows. As with Algorithms 6.4 and 6.5, for each query condition, $x_i \in X$, if an interval, I, is a contributing interval for x_i, then I is *also* a contributing interval for the next query condition, x_{i+1}. Moreover, according to the weighted approach, if an interval, I, is a *conservative* contributing interval for two or more query conditions, then I contributes *equally*, i.e., with the probability of I, to the result of each probability query given by those query conditions. However, also according to the weighted approach, if an interval, I, is a *non-conservative* contributing interval for two or more query conditions, then I may contribute *differently*, i.e., with different probabilities, to the result of each probability query given by those query conditions. For these reasons, we need to consider each *conservative* contributing interval in P_e only once, and in the algorithm, we maintain a single probability running total for conservative contributing intervals, p_c (see line 3). However, we need to consider other intervals once per query condition. For this reason, in the algorithm, we have two loops, one that iterates through query conditions (see line 5) and another one that iterates through intervals (see line 7). These intervals are contained in the count distribution R, which at the very beginning is a copy of P_e (see line 4). For each query condition, $x \in X$, there is one iteration of the loop in line 5. When $(k-1)$ iterations have finished, we may have considered some of the *conservative* contributing intervals of the k^{th} query condition, p_c is the sum of probabilities of these intervals, and R contains the remaining sequence of the contributing intervals of the k^{th} query condition. During the k^{th} iteration, we first set the running total for the k^{th} query condition, p_x, to p_c (see line 6) and then we go through the remaining sequence of its contributing intervals (see the loop in line 7) until we reach the end of the sequence, i.e., until the loop condition fails. For each encountered interval, we add the probability, with which the interval contributes to the result, to the running total for the current query condition, p_x (see lines 9 and 13). In addition, if the encountered interval is a *conservative* contributing interval (see line 8), we remove the interval from R (line 10) and add its probability to the running total for conservative intervals, p_c (see line 11) When we reach the end of the sequence, we insert the pair of a query condition and its probability, (x, p_x), into the set of sampled points of the approximation of the CDF, F_e (see line 14). The algorithm terminates when the running probability total, p_x, reaches 1 (see line 15), which happens when all the query conditions have been considered or when all the intervals from P_e are contributing intervals for the remaining, unconsidered query conditions.

7 Experiments

The methods for using and creating pre-aggregated data described in Sections 4 and 5 are implemented by a prototype system. The implementation is based on the Java implementation of An Incomplete Data Cube [1, 7]. In this section, we report on the results of initial experiments performed with the prototype. The experiments were performed on a desktop machine with the 1.80 GHz Intel Pentium 4 CPU and 1 GB RAM, running Windows 2000 OS. The code was compiled and run on Sun Java 2 Platform, Standard Edition 1.4.2. The database management system used was Sleepycat Berkeley DB 4.2.52.

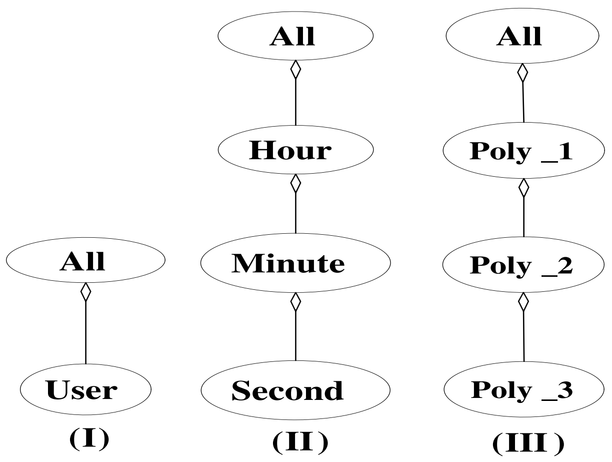


Figure 11: Simplified dimension types $\mathcal{T}_u$ (I), $\mathcal{T}_t$ (II), and $\mathcal{T}_r$ (III) used for experiments

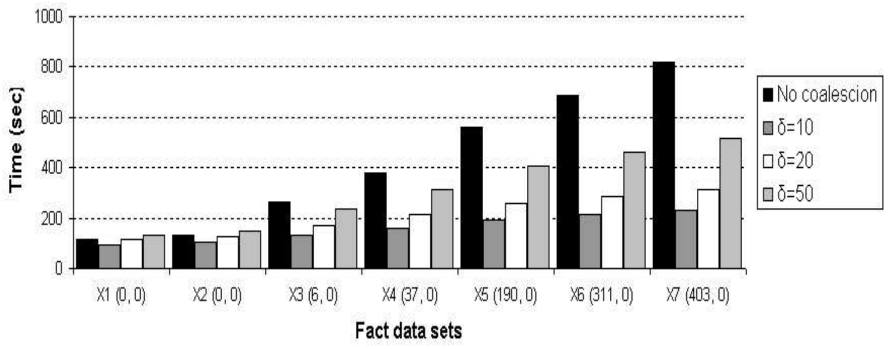
7.1 Data

In our experiments, we use a three-dimensional multidimensional object. It consists of the USER, TIME, and LOCATION dimensions. The schema of this multidimensional object is a simplified version of the schema of the object from Section 3. The types of the dimensions can be seen in Figure 11. The types $\mathcal{T}_u$, $\mathcal{T}_t$, and $\mathcal{T}_r$ describe the USER, TIME, and LOCATION dimensions, respectively. In the USER dimension, a dimension value from the category *User* is a user ID. The TIME dimension is a standard one. The LOCATION dimension is a hierarchical representation of the city of Oldenburg, Germany: a dimension value from the *Poly_1* category is an edge from the graph representation of the city (see [12]), a dimension value from the *Poly_2* category is a line from the 2D representation of the city (see [12]), and the *Poly_3* category is obtained from the 2D representation by dividing its lines into smaller segments. The *Poly_1*, *Poly_2*, and *Poly_3* category contains approximately 4,000, 7,000, and 68,000 dimension values, respectively.

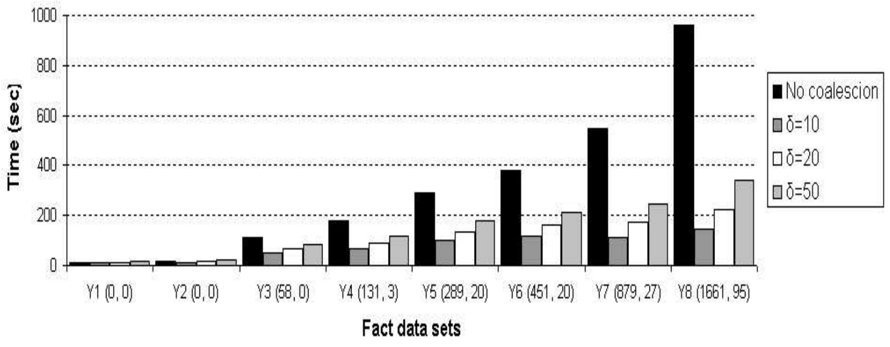
The fact data for experiments is created as follows. First, by using a modified version of Brinkhoff's generator [3], we simulate and record GPS readings of positions of 150,000 vehicles moving in the city of Oldenburg, Germany for 5 hours. Then, we select one *current time point*. For this current time point, t, we select several *future time points*. Then, given the information about vehicle positions at t, we predict one set of positions of the vehicles with their probabilities at each selected future time point. We use a simple prediction algorithm that, given the current position of a vehicle, finds all the segments (from the 2D representation of Oldenburg, i.e., from the *Poly_2* category) that the vehicle can reach under the assumption that its future speed does not exceed its maximum speed and speed limits on the segments. All the reachable segments receive the same probability to be reached. Thus, a predicted vehicle position is a segment.

Each set of predicted future time positions is used to create a fact data set. For this, a predicted vehicle position is transformed into a probabilistic relationship between a fact and the LOCATION dimension and relationships between that fact and the two other, USER and TIME, dimensions. For all data sets, all facts are related to the *User* category in the USER dimension and to the *Second* category in the TIME dimension. In a fact data set, all facts are related to the same category from the LOCATION dimension, which is either the *Poly_2* or *Poly_3* category. Specifically, each vehicle position is transformed as followed. Suppose we predict that a vehicle, f, will be on a segment e with the probability p. Then, if we create a fact data set, where facts are related to *Poly_2* category, we create a fact-dimension relationship, (f, e, p). However, if we create a fact data set, where facts are related to *Poly_3* category, we create a set of fact-dimension relationships, $\{f, e_i, \frac{p}{N}, i = 1, \ldots, N\}$, where N is the number of children of e. In total, we create 7 fact data sets for the *Poly_3* category and 8 fact data sets for the *Poly_2* category. We term the former series of fact data sets as the *Poly_3 series* and the latter series of fact data sets as the *Poly_2 series*. In addition to these fact data sets, we create 5 fact data sets for different current time points and future time points. The facts in these fact data sets are related to the *Poly_3* category according to the procedure as for the *Poly_3* series. We term this series of fact data sets as the *very complex series*. The meaning of the name of this series is explained shortly. Thus, in total we create 7+8+5 = 20 fact data sets. We denote the fact data sets from the *Poly_3* series by X1, X2, ..., X7, from the *Poly_2* series by Y1, Y2, ..., Y8, and from the *very complex series* by Z1, Z2, ..., Z5.

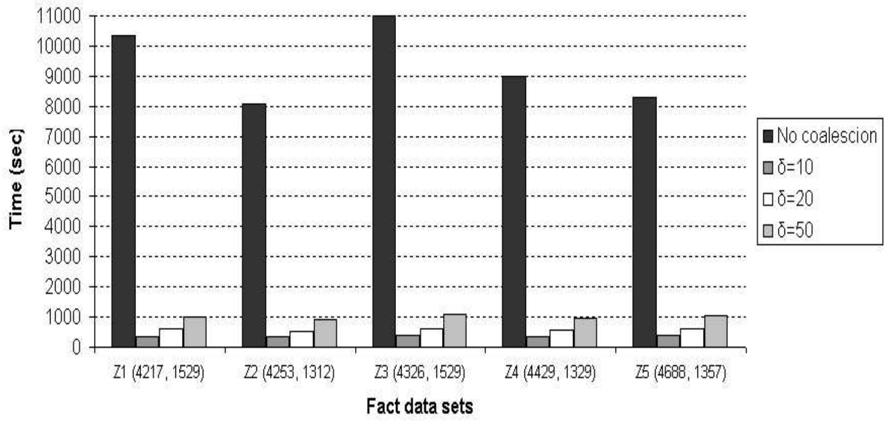
We define *complexity* of a fact data set as the number of dimension values such that the number of facts attached to these values is greater than a given threshold. It is natural to choose a threshold that is greater than the average number of facts per one dimension value. In fact, this is equivalent to counting number of distributions in the result of the pre-aggregation procedure *without* coalescion such that the size of the distributions is greater than a given threshold. If a fact data set is not complex, we call it *simple*. For example, if we set the threshold to 200, then the complexity of our fact data sets are as follows. For the *Poly_2* and *Poly_3* series, the number of the dimension values to which more than 200 facts are related, per fact data set, is between 0 and 403 (out of 7,000) and



(a)



(b)



(c)

Figure 12: Average time complexity of computing probability distributions, for the (a) *Poly_3*, (b) *Poly_2*, and (c) *very complex* series

between 0 and 1661 (out of 68,000), respectively. The *very complex* series is much more complex. For this series, the number of the dimension values to which more than 200 facts are related, per fact data set, is between 4217 and 4688 (out of 68,000). The numbers of relationships between facts and the LOCATION dimension per fact data set are (1) from 150,000 to 1,040,000, for the *Poly_2* series, (2) from 1,453,000 to 3,135,000, for the *Poly_3* series, and (3) from 3,478,000 to 4,190,000, for the *very complex* series. Note that the maximum size of a fact data set from the *Poly_3* series is not much different from the sizes of fact data sets from the *very complex* series. However, the complexity of the two series is very different. Given a data set, the average probability of the relationships between facts and dimension values from the LOCATION dimension in that data set is between 0.036 and 0.998.

7.2 Computing Pre-Aggregated Probability Distributions

By the basic method from Section 5.1, each fact data set is transformed into a set of pre-aggregated count distributions for the categories *All* in the USER and TIME dimension and the category from the LOCATION dimension to which the facts are attached. Each fact data set is transformed once for a combination of values of the following parameters: (1) $\delta = 10, 20, 50$ (for the definition, see Section 4.2), (2) $m_\delta = 0.25 \cdot \delta, 0.5 \cdot \delta, 0.75 \cdot \delta$ (for the definition, see Section 4.3), and (3) $\rho = 0, 1, 2$ (for the definition, see Section 4.3). In addition, each fact data set is transformed once without coalescion. Thus, we create a collection of $3 \cdot 3 \cdot 3 + 1 = 28$ sets of pre-aggregated count distributions per fact data set. Thus, in total, we create $20 \cdot 28 = 560$ sets of pre-aggregated count distributions.

Figures 12(a), 12(b), and 12(c) show the average time taken by the pre-aggregation procedure for each fact data set from the *Poly_3*, *Poly_2*, and *very complex* series, respectively. Specifically, the figures compare the time complexity of the pre-aggregation procedure *without* and *with* coalescion (using different values of δ). In the figures, the horizontal axis is labeled with fact data set identifiers (e.g., X1). Our experiments show that the time complexity of our pre-aggregation procedure, which is based on the convolution, depends a lot on the complexity of fact data sets. For this reason, the pair of numbers next to an identifier describes the complexity of the corresponding fact data set. Specifically, the first and second number from the pair is the number of the dimension values to which more than 200 and 500 facts are related, respectively. On the horizontal axis, the fact data sets are sorted by the first number of the pair in ascending order. For the *Poly_3* and *Poly_2* series, this sorting corresponds to the sorting by the time complexity. This is not true for the *very complex series*, because in that series time complexity is highly influenced by the second number from the pair, i.e., by the number of the dimension values to which more than 500 facts are related. For example, that is why the pre-aggregation procedure is more time-consuming for the fact data set Z1 than for the fact data set Z5. In addition, we observe that generally the time complexity of the pre-aggregation procedure is somewhat higher for the *Poly_3* series than for the *Poly_2* series. This is because the fact data sets in the former series are much larger than in the latter series and because the number of dimension values in the *Poly_3* category is much greater than in the *Poly_2* category (see Section 7.1). At the same time, we emphasize that despite the great difference in sizes of fact data sets, the time complexity of the pre-aggregation procedure for the *Poly_2* and *Poly_3* series are *comparable*. However, the time complexity of the pre-aggregation procedure for the *very complex* series is much higher than the time complexity of the pre-aggregation procedure for the *Poly_3* series, although the sizes of fact data sets for the two series are *comparable*. Thus, we conclude that the time complexity of the pre-aggregation procedure depends more on complexity of fact data sets than on their sizes.

In general, we conclude that complementing the pre-aggregation procedure with coalescion significantly reduces the time complexity for almost any fact data set. The four exceptions are the fact data sets X1 and X2 (see Figure 12(a)) and Y1 and Y2 (see Figure 12(b)). We believe that for these fact data sets coalescion is not effective, because the sets are simple. Specifically, there are no dimension values with more than 200 and 500 facts from these sets. This means that even without coalescion, only a small number of *long* distributions, i.e., the distributions that contain more than 50 intervals, are produced and most of the produced distributions are *short*, i.e., contain less than 50 intervals. For example, for the fact data set X1 only 356 out of 68042 produced distributions are long, compared to 12108 long distributions for the fact data set X4. For this reason, coalescion with the considered values of δ, i.e., 50 and less, does not significantly reduce the time complexity of the pre-aggregation procedure for the fact data sets X1, X2, Y1, and Y2. Moreover, for these fact data sets, with $\delta = 50$, the convolution with coalescion is more expensive than without coalescion. However, smaller values of δ can be used to make coalescion more effective.

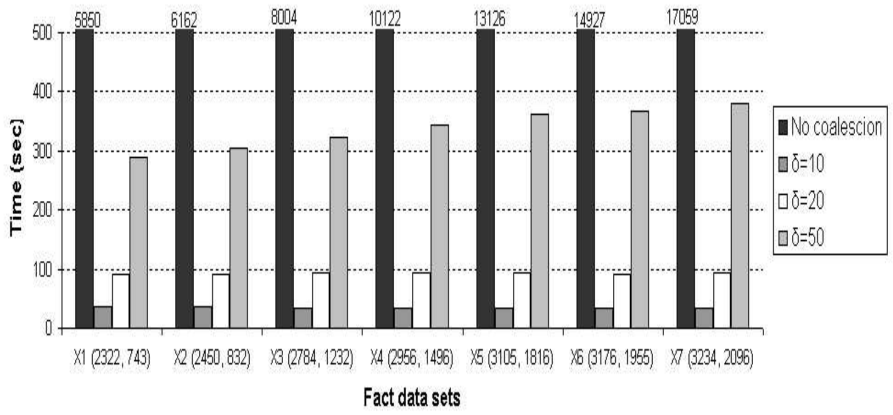
Furthermore, the effectiveness of coalescion grows as the complexity of the fact data sets increases, for all the three series. Let us first consider the *Poly_3* series. In Figure 12(a), for the fact data set X3, the convolution with coalescion with the values of δ of 10, 20, and 50 takes, respectively, 50%, 65%, and 89% of the time needed for the convolution without coalescion. For comparison, for a more complex fact data set X4, the convolution with coalescion with the values of δ of 10, 20, and 50 takes, respectively, 42%, 56%, and 82% of the time needed for the convolution without coalescion. Moreover, for the most complex fact data set X7, the convolution with coalescion with the values of δ of 10, 20, and 50 takes, respectively, 28%, 38%, and 62% of the time needed for the convolution without coalescion. Similar trend of the growing coalescion effectiveness can be noticed for the *Poly_2* series, in Figure 12(b). For the *very complex* series of fact data sets (see Figure 12(c)), the pre-aggregation procedure without coalescion is extremely time consuming, compared to the other two series. This is because the *very complex* series is much more complex than the other two series. For this series, the effectiveness of coalescion is even more significant. For example, for the fact data set Z3, the convolution with coalescion with the values of δ of 10, 20, and 50 takes, respectively, 3%, 6%, and 10% of the time needed for the convolution without coalescion. Thus, the experiments confirm our hypothesis that the pre-aggregation procedure based on the *Count* algorithm from Algorithm 4.1, is hardly scalable (in terms of the complexity of fact data sets) *without* coalescion, but becomes much more scalable when complemented *with* coalescion.

As for the usability of our pre-aggregation procedure in terms of the time complexity, it may be argued that the current prototype implementation may be not fast enough for even simple, but large data sets. For example, a simple fact data set X2 contains 1,542,000 fact-dimension relationships. However, the pre-aggregation procedure with $\delta = 10$ for this fact data set takes 102 seconds. This is arguably too slow for the real-world location-based services that answer queries for *hyper-dynamic* data, e.g., moving cars. Query results may become outdated before computations are complete. A way to improve time complexity is to decrease the value of δ. However, one can argue that the lower the value of δ, the worse the precision (see "Precision" in Section 7.3). Anyway, we emphasize that this is a *prototype* implementation, but acknowledge that further optimizations to our pre-aggregation procedure are needed if it is going to be used in the real-world location-bases services. At the same time, our pre-aggregation procedure can be applied to the probabilistic data of any degree of dynamism and the current implementation may be fast enough for queries for *slowly changing* data, e.g., historical data.

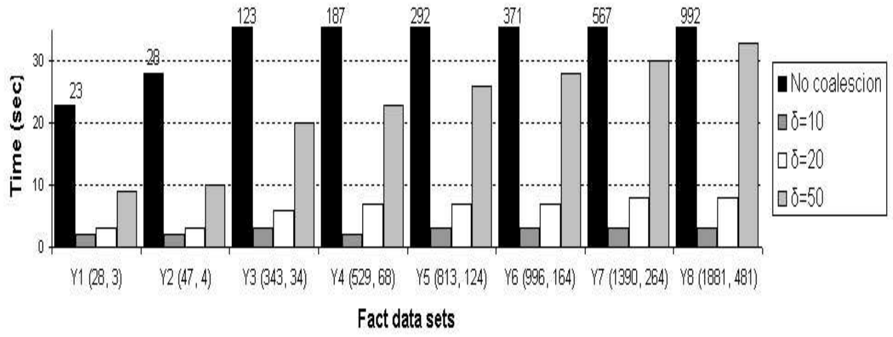
7.3 Using Pre-aggregated Probability Distributions for Aggregation

In Section 7.2, we produce sets of pre-aggregated count distributions. In this section, these sets are used for computing sets of aggregate values (i.e., sets of count distributions) by the method from Section 4. We aggregate to the *Poly_1* category. A set of pre-aggregated count distributions is aggregated once with the parameter (δ, m_δ, and ρ) values inherited from the pre-aggregation procedure that created that set (see Section 7.2). We report on the experiment results for every set of pre-aggregated count distributions except those mentioned in the following. When used without coalescion, our prototype system runs out of memory for the sets of pre-aggregated count distributions produced from the fact data sets from the *very complex* series. The reason for that is not yet determined. Consequently, we do not report on the time complexity for those sets, when used without coalescion.

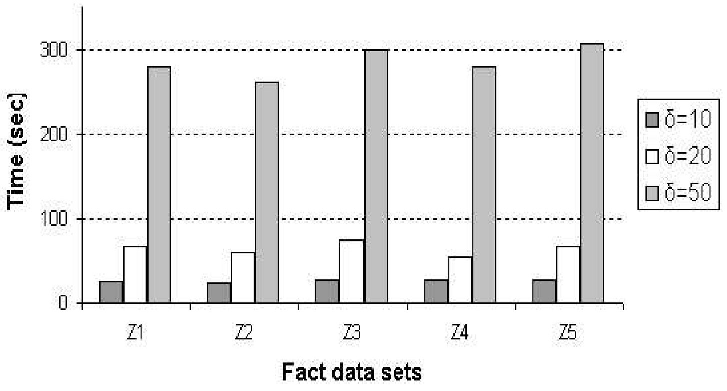
Time Complexity In the following, we measure time complexity of the aggregation procedure. We refer to a collection of sets of pre-aggregated count distributions by the identifier of the fact data set from which the collection is produced. For example, a collection of sets of pre-aggregated count distributions X1 is produced from the fact data set X1. In addition, we refer to a series of collections of sets of pre-aggregated count distributions by the name of the corresponding series of fact data sets. For example, the *Poly_2* series of collections of sets of pre-aggregated count



(a)



(b)



(c)

Figure 13: Average time complexity of using pre-aggregated count distributions for aggregations, for the (a) *Poly_3*, (b) *Poly_2*, and *very complex* series

distributions is produced from the *Poly_2* series of fact data sets. The definition of *complexity* of sets of pre-aggregated count distributions is analogous to the definition of complexity of fact data sets. We define *complexity* of a set of pre-aggregated count distributions as the number of the distributions in the result of the aggregation procedure *without* coalescion such that the size of the distributions is greater than a given threshold. If a set is not complex, we call it *simple*. For example, if we set the value of threshold to 200, the complexity of our sets of pre-aggregated count distributions is as follows. For the *Poly_3* and *Poly_2* series, the number of the distributions longer than 200 is between 2322 and 3234 and between 28 and 1881, respectively. The complexity of the very complex series is not yet determined.

Figures 13(a), 13(b), and 13(c) show the average time taken by the aggregation procedure for each set of pre-aggregated count distributions from the *Poly_3*, *Poly_2*, and *very complex* series, respectively. Specifically, the figures compare the time complexity of the aggregation procedure *without* and *with* coalescion (using different values of δ). In the figures, the horizontal axes are labeled with identifiers of the collections of the sets of pre-aggregated count distributions used for aggregation. Analogously to the case of the pre-aggregation procedure, our experiments show that the time complexity of our aggregation procedure, which is based on the convolution, depends on a lot on the complexity of sets of pre-aggregated count distributions. For this reason, in Figures 13(a) and 13(b), the pair of numbers next to an identifier describes the complexity of the corresponding set of pre-aggregated count distributions. Specifically, the first and second number from the pair is the number of distributions (in the result of the aggregation procedure *without* coalescion) longer than 200 and 500, respectively. On the horizontal axis, the collections of sets of pre-aggregated count distributions are sorted by the first number of the pair in ascending order. For the *Poly_3* and *Poly_2* series, this sorting corresponds to the sorting by the time complexity. In Figures 13(a) and 13(b), the bars for run times of the aggregation procedure *without* coalescion are cut, because they are two long to be shown. Instead, a number over a bar indicates the value that corresponds to this bar. As in the case of the pre-aggregation procedure (see Section 7.2), we observe that generally the time complexity of the aggregation procedure is higher for the *Poly_3* series than for the *Poly_2* series. The time complexity is even higher for the *very complex* series. We believe that this is primarily due to the difference in complexity of the series. (We expect that the sets of pre-aggregated count distributions in the *very complex* series would be much more complex than the sets from the other two series). However, we believe that the difference in sizes of the sets of pre-aggregated count distributions also contributes to the difference in time complexity.

We conclude that complementing the aggregation procedure with coalescion significantly reduces the time complexity for any set of pre-aggregated count distributions. Unlike with the pre-aggregation procedure, there are no exceptions, because the count distributions produced by the aggregation procedure are long enough for coalescion with the considered values of δ to be effective. Most notably, as mentioned in Section 7.2, the sets of pre-aggregated count distributions produced *without* coalescion from the collections X1, X2, Y1, and Y2 contain a small number of *long* distributions (i.e., those distributions that contain more than 50 intervals). However, when used for aggregation, these sets produce many long count distributions. For example, for the set from the collection X1 as many as 3657 out of 3775 produced count distributions are long.

As in the case of *pre-aggregation* (see Section 7.2), the effectiveness of coalescion grows as the complexity of the sets of pre-aggregated count distributions increases, for both the *Poly_3* and *Poly_2* series. Let us first consider the *Poly_3* series. In Figure 13(a), for example, for the set X1, the convolution with coalescion with the values of δ of 10, 20, and 50 takes, respectively, 0.6%, 1.5%, and 5.0% of the time needed for the convolution without coalescion. One can see from the figure that as the complexity of the sets grows, the time complexity of the convolution *without* coalescion grows significantly, while the time complexity of the convolution *with* coalescion grows very little (e.g., it is almost constant for the values of δ of 10 and 20). For comparison with the set X1, for the most complex fact data set X7, the convolution with coalescion with the values of δ of 10, 20, and 50 takes, respectively, 0.2%, 0.6%, and 2.2% of the time needed for the convolution without coalescion. Similar trend of the growing coalescion effectiveness can be noticed for the *Poly_2* series, in Figure 13(b). Thus, as in Section 7.2, the experiments confirm our hypothesis that the aggregation procedure based on the *Count* algorithm from Algorithm 4.1, is hardly scalable *without* coalescion (in terms of complexity of pre-aggregated count distributions), but becomes much more scalable when complemented *with* coalescion.

As for the usability of our aggregation procedure in terms of the time complexity, it may be argued that the current prototype implementation may be not fast enough for complex sets of the pre-aggregated count distributions. For example, while the aggregation procedure *with* coalescion is fast enough for the *Poly_2* series, i.e., it takes less than 5 seconds with $\delta = 10$ for every set of pre-aggregated count distribution, the procedure is much slower for the *Poly_3* series, i.e., it takes around 35 seconds with $\delta = 10$ for every set of pre-aggregated count distributions. As mentioned before, this may be too slow for the real-world location-based services that answer queries for *hyper-dynamic* data, e.g., moving cars. Query results may become outdated before computations are complete. A way to improve time complexity is to decrease the value of δ. However, one can argue that the lower the value of δ, the worse the precision (see "Precision" in Section 7.3). Anyway, we again emphasize that this is a *prototype* implementation, but acknowledge that further optimizations to our pre-aggregation procedure are needed if it is going to be used in the real-world location-bases services. At the same time, our aggregation procedure can be applied to the probabilistic data of any degree of dynamism and the current implementation may be fast enough for queries for *slowly changing* data, e.g., historical data.

Precision In the following, we measure the quality of count distributions produced with coalescion in terms of their difference from aggregate values produced without coalescion. The measure of the quality is the *distance* between an interval from a non-coalesced count distribution and a corresponding coalesced count distribution. This is formalized in Definition 7.1.

Definition 7.1. [**Distance between an interval and a count distribution**] Given an interval, I, we denote its length by $|I|$. Given two intervals, I and J, we denote their intersection by $|I \cap J|$. Given a number, r, we denote its absolute value by $|r|$. Then, given a dimension value $e \in C_{to}$, an interval, (I, p), and a count distribution for e,

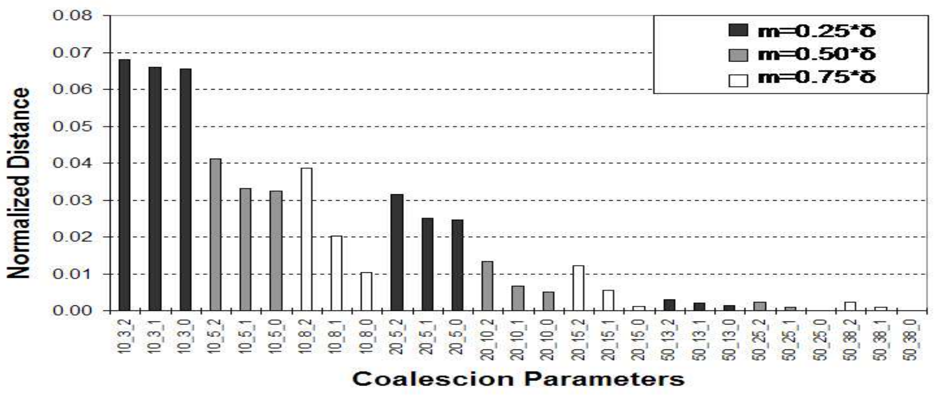
$$Count(e) = \{(J_1, q_1), (J_2, q_2), \ldots, (J_m, q_m)\},$$

the *distance* between I and $Count(e)$, $dist(I, Count(e))$, is defined as follows:

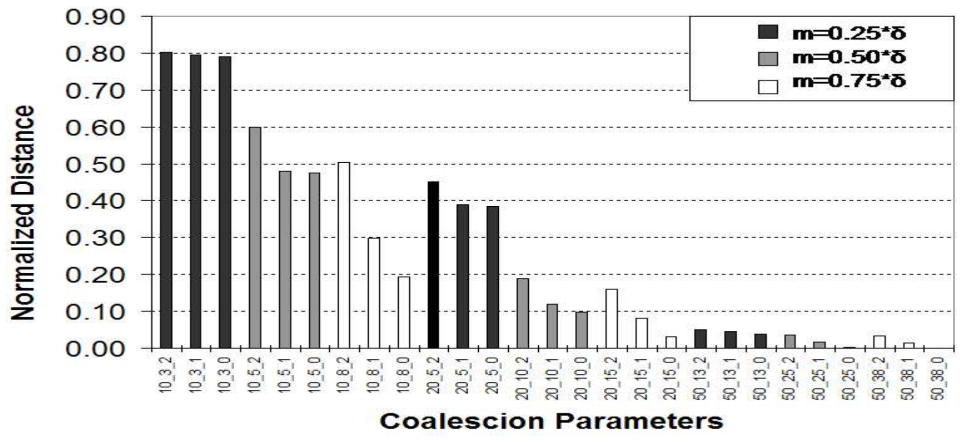
$$dist(I, Count(e)) = |p - \sum_{i=1}^{m} \frac{q_i \times |I \cap J_i|}{|J_i|}|$$

In addition, $dist_N(I, Count(e)) = \frac{dist(I, Count(e))}{p}$ is called *normalized distance* between I and $Count(e)$. ∎

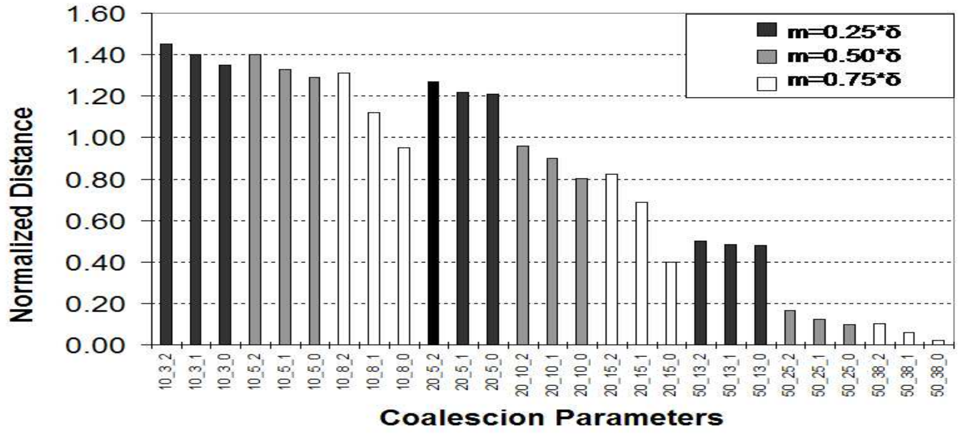
In the following, we explain the semantics of the distance. If pre-aggregated count distributions are computed *without* coalescion, then count distributions computed from this pre-aggregated data also *without* coalescion will contain non-overlapping intervals only. Suppose (I, p) is an interval from a non-coalesced count distribution that contains only non-overlapping intervals, $Count_1(e)$, and $Count_2(e)$ is a coalesced version of $Count_1(e)$. Next, given $I = [a; b]$, suppose Q is the following probability query (see Section 6.2): "What is the probability that the count for e is between a and b, both numbers included?". If we pose the query Q to $Count_1(e)$ and $Count_2(e)$ and process the query according to the weighted approach, the responses to the query are the numbers p and $\sum_{i=1}^{m} \frac{q_i \times |I \cap J_i|}{|J_i|}$, respectively. Thus, $dist(I, Count_2(e))$ is the *absolute* difference in responses to the query Q obtained by using the precise distribution, $Count_1(e)$, or the approximated distribution, $Count_2(e)$, according to the weighted approach. In addition, the corresponding normalized distance, $dist_N(I, Count_2(e))$, shows the *relative difference*, i.e., how large the difference is, compared to the probability of the interval, p. Specifically, if $dist_N(I, Count_2(e)) < 1$,



(a)



(b)



(c)

Figure 14: Average normalized distance, for *highly certain* (a), *moderately uncertain*(b), and *highly uncertain*(c) fact data sets

then the difference is less than p and can be considered small. Otherwise, the difference can be considered large. Thus, the most natural way to measure precision is to perform experiments with the query processing techniques (see Sections 6.2 and 6.3). However, because the techniques are not implemented yet, we use the distance notion instead.

We denote by AFP average probabilities of the relationships between facts and dimension values from the LOCATION dimension. Figures 14(a), 14(b), and 14(c) present average normalized distance for three collections of fact data sets, i.e., for *highly certain* ($AFP \in [0.914, 0.998]$), *moderately uncertain* ($AFP \in [0.24, 0.44]$), and *highly uncertain* ($AFP \in [0.048, 0.19]$) fact data sets, respectively. The *highly certain* and *moderately uncertain* fact data sets constitute a part of the *Poly_2* series and the *highly uncertain* fact data sets cover completely the *Poly_3* and include a part of the *Poly_2* series. (The *very complex* series is not used, because we could not produce non-coalesced count distributions from these series and, consequently, could not compute the distance.) The distance is presented for each combination of the coalescion parameters. Thus, in the horizontal axis, 10_3_2 means $\delta = 10$, $m_\delta = 3$, and $\rho = 2$. The combinations of the coalescion parameters are sorted so that given two combinations, it is expected that the combination that lies closer to the right edge of the horizontal axis will yield better precision, i.e., smaller ratio between the distance values and interval probabilities. Specifically, the combinations that contain the low value of δ, i.e., $\delta = 10$, are placed at the left edge of the horizontal axis. These combinations are followed by those that contain the average value of δ, i.e., $\delta = 20$, and, in turn, by those that contain the high value of δ, i.e., $\delta = 50$. This means that there are three *large* groups of the combinations, one per value of δ. Inside each large group, the combinations are further grouped into *small* groups according to the value of m_δ, and the small groups for $m_\delta = 0.75 \cdot \delta$ are followed by those for $m_\delta = 0.50 \cdot \delta$, which are, in turn, followed by those for $m_\delta = 0.75 \cdot \delta$. Inside each small group, the combination for $\rho = 2$ is followed by the one for $\rho = 1$, which is, in turn, followed by the one for $\rho = 0$.

In general, Figures 14(a), 14(b), and 14(c) show that the coalesced count distributions are highly precise for highly certain and moderately uncertain data sets (the average normalized distance is always less than 1 for every value of δ). For highly uncertain data sets, the coalesced count distributions are quite precise for a high value of δ, i.e., 50 (the average normalized distance is always less than 1), but becomes imprecise for lower values of δ, i.e., 10 and 20 (the average normalized distance is in many cases greater that 1). However, the coalescion parameters help significantly improve the precision for every data set in almost any case. The level of precision is generally determined by δ, so we recommend increasing the value of δ as the first measure to increase the precision. For the same values of δ, setting higher values of m_δ help increase the precision even further. For the same values of δ and m_δ, setting higher values of ρ is effective (if m_δ is high enough). The reasons for these effects of the parameters are discussed in Section 7.4.

7.4 Examples of Count Distributions

In addition to discussing the quality of sets of coalesced count distributions as a whole, in Examples 7.2 and 7.3, we present several concrete count probability distributions that are produced during the experiments, exactly as they appear in our logs.

Example 7.2. [**Non-coalesced count distribution**] In this example, we present a precise, non-coalesced count distribution, which can be seen below. Note that the distribution contains many intervals with extremely low probabilities (e.g., 58 out of 86 intervals have a probability of less than 0.001). Suppose processing a user query without coalescion involves producing this distribution. If the system predicts that it would take too much time and/or space to produce the distribution, then given such a high precision, before processing the query the system may ask a user whether the user agrees to trade some precision for time and space complexity by turning on coalescion.

$([0;0], 3.179347095886113E-10)([1;1], 8.234264083616268E-9)([2;2], 1.0498858168375834E-7)$

$([3;3], 8.78506401986354E-7)([4;4], 5.426257120777551E-6)([5;5], 2.6384542001703962E-5)$

$([6;6], 1.0517962988638688E-4)([7;7], 3.5349987840882724E-4)([8;8], 0.0010223115485663)$

$([9;9], 0.002583787033489785)([10;\ 10], 0.005777047010525943)([11;\ 11], 0.011539696985985056)$

$([12;\ 12], 0.0207597701956348)([13;\ 13], 0.03386181896648201)([14;\ 14], 0.0503646470574778)$

$([15;\ 15], 0.06864017702127574)([16;\ 16], 0.08607706473551144)([17;\ 17], 0.09968607508404385)$

$([18;\ 18], 0.10695557212303336)([19;\ 19], 0.10661341369707565)([20;\ 20], 0.09897742683265986$

$([21;\ 21], 0.08576921877004552)([22;\ 22], 0.06951009268028394)([23;\ 23], 0.05277690199853426)$

$([24;\ 24], 0.03760082907587592)([25;\ 25], 0.025171837758547134)([26;\ 26], 0.015854011618459413)$

$([27;\ 27], 0.009404933663079816)([28;\ 28], 0.005260154580178331)([29;\ 29], 0.002776223271920063)$

$([30;\ 30], 0.0013837808902744221)([31;\ 31], 6.518411008318314E-4)([32;\ 32], 2.90365809700994E-4)$

$([33;\ 33], 1.2238053880813055E-4)([34;\ 34], 4.8825237762428826E-5)([35;\ 35], 1.844647942918667E-5)$

$([36;\ 36], 6.601815021705062E-6)([37;\ 37], 2.238775445949463E-6)([38;\ 38], 7.195247639545527E-7)$

$([39;\ 39], 2.1919707066571283E-7)([40;\ 40], 6.330224700185138E-8)([41;\ 41], 1.7330732073742502E-8)$

$([42;\ 42], 4.498036718575981E-9)([43;\ 43], 1.1066479420594533E-9)([44;\ 44], 2.580630352503483E-10)$

$([45;\ 45], 5.70293957954205E-11)([46;\ 46], 1.1940773547110665E-11)([47;\ 47], 2.3681431710976016E-12)$

$([48;\ 48], 4.447161841570579E-13)([49;\ 49], 7.904804666438235E-14)([50;\ 50], 1.329357116514118E-14)$

$([51;\ 51], 2.1140635789370482E-15)([52;\ 52], 3.177438870555375E-16)([53;\ 53], 4.510743778815912E-17)$

$([54;\ 54], 6.044083961935451E-18)([55;\ 55], 7.638226745794614E-19)([56;\ 56], 9.096395978631058E-20)$

$([57;\ 57], 1.019906047527562E-20)([58;\ 58], 1.075537548396446E-21)([59;\ 59], 1.065575438477942E-22)$

$([60;\ 60], 9.906237941596954E-24)([61;\ 61], 8.630211973197655E-25)([62;\ 62], 7.035397789366963E-26)$

$([63;\ 63], 5.358144218264031E-27)([64;\ 64], 3.805685405541336E-28)([65;\ 65], 2.5159348833680943E-29)$

$([66;\ 66], 1.5448253980439302E-30)([67;\ 67], 8.7889167243569E-32)([68;\ 68], 4.620729562892407E-33)$

$([69;\ 69], 2.2382535409044147E-34)([70;\ 70], 9.955704032510477E-36)([71;\ 71], 4.050829500271661E-37)$

$([72;\ 72], 1.5011891837966789E-38)([73;\ 73], 5.0416121191856066E-40)([74;\ 74], 1.5255033504289756E-41)$

$([75;\ 75], 4.130333555513952E-43)([76;\ 76], 9.924912573672056E-45)([77;\ 77], 2.0956528059645797E-46)$

$([78;\ 78], 3.84074329465698E-48)([79;\ 79], 6.014832389558719E-50)([80;\ 80], 7.885677133389753E-52)$

$([81;\ 81], 8.415471470037632E-54)([82;\ 82], 7.01885073811092E-56)([83;\ 83], 4.2897060360151254E-58)$

$([84;\ 84], 1.7079176873149942E-60)([85;\ 85], 3.322797088737615E-63)$

Example 7.3. [**Coalesced count distributions**] In this example, we present a scenario that a system could follow to first obtain a compact approximation of the count distribution from Example 7.2 and then to refine the approximation at the user's request. Specifically, we present a sequence of coalesced versions of the count distribution from Example 7.2. The coalesced count distributions are enumerated below, together with the values of the coalescion parameters used. The comments to the distributions can be used as guidelines for selecting values of the coalescion parameters.

1. $\delta = 10$, $m_\delta = 3$, $\rho = 1$:

   $([0;\ 70], 0.14396207208736794)([4;\ 59], 0.11010743052854795)([5;\ 60], 0.036702476842849316)([5;\ 65], 0.12198835895742445)$

   $([6;\ 66], 0.04066278631914148)([6;\ 70], 0.13289740466965425)([7;\ 71], 0.044299134889884746)([8;\ 75], 0.16353757013061587)$

   $([10;\ 84], 0.1897758117076795)([12;\ 85], 0.016066953866834446)$

   The above distribution is obtained when a system sets a relatively low value of δ, 10. The distribution is much more compact than the one from Example 7.2. However, the distribution does not contain any short "good" intervals, which may cause low precision of query answers. A possible reason for this is that with a lower value of δ larger groups of intervals are coalesced, so there are less short intervals. As stated in Section "Precision", the system should increase the value of δ, which will produce smaller groups (and will also allow higher values of m_δ), to obtain a significant increase in precision.

2. $\delta = 20$, $m_\delta = 5$, $\rho = 1$:

   $([0; 51], 0.07092523494851453)([1; 54], 0.06777861075672777)([2; 55], 0.06801741961456939)([3; 60], 0.08261516105955388)$

   $([4; 64], 0.08211651387591874)([15; 15], 0.014464304439321583)([16; 16], 0.0321551470097693)([16; 58], 0.01585807129617124)$

   $([17; 17], 0.04299201627141359)([17; 63], 0.04302602989217311)([18; 18], 0.0408886590518133)([18; 68], 0.07142046175223914)$

   $([19; 19], 0.01917054019977402)([19; 71], 0.09598321207423986)([20; 76], 0.10313978041114732)([21; 81], 0.08513261204785198)$

   $([22; 85], 0.06431622529880124)$

   After the system increased the value of δ from 10 to 20 (and the value of m_δ increased proportionally from 3 to 5), the above distribution is obtained. Compared to the previous distribution, this distribution is more precise, e.g., it contains 5 short "good" intervals, the sum of their probabilities is high enough (0.15), and the rest of the intervals are significantly shorter. This exemplifies the transfer to a different level of precision when increasing the value of δ, which is seen in Figure 14(b) and noted in Section "Precision". As also stated in Section "Precision", the system could gain even more precision by increasing the value of m_δ.

3. $\delta = 20$, $m_\delta = 15$, $\rho = 1$:

   $([0; 82], 0.11897551078750496)([11; 11], 0.0050097110453993)([12; 12], 0.014848002237739388)([13; 13], 0.028369767499965812)$

   $([14; 14], 0.04523795660429487)([14; 83], 0.009359527744887767)([15; 15], 0.06400274385506959)([16; 16], 0.08196238732016445)$

   $([17; 17], 0.0959204252406539)([17; 84], 0.005453297465745496)([18; 18], 0.10314499676472423)([19; 19], 0.10222618755689492)$

   $([19; 85], 4.357914120983518E-4)([20; 20], 0.09348505135493608)([21; 21], 0.07876158425783535)([22; 22], 0.060744059195773)$

   $([23; 23], 0.042314888692954654)([24; 24], 0.025788462868168216)([24; 38], 0.012468771961465078)([25; 25], 0.011490876133724)$

   After the system increased the value of m_δ from 5 to 15, the above distribution is obtained. Compared to the previous distribution, this distribution is even more precise, e.g., it contains 15 short "good" intervals instead of 5 and the sum of their probabilities is 0.85 instead of 0.15. Compared to the previous distribution, the rest of the intervals are longer, but the sum of their probabilities is much lower. This exemplifies the extreme importance of being able to increase the value of m_δ, which again exemplifies the transfer to a different level of precision when increasing the value of δ. As discussed in Section "Precision", the system could gain even more precision by decreasing the value of ρ.

4. $\delta = 20$, $m_\delta = 15$, $\rho = 0$:

   $([0; 82], 0.03063417635861396)([11; 11], 0.006001870967576094)([12; 12], 0.017275615631781215)([13; 13], 0.0318897122101724)$

   $([14; 14], 0.049292155375020484)([15; 15], 0.06809617767823117)([16; 16], 0.08582115045832502)([16; 83], 0.00366132338593119)$

   $([17; 17], 0.09957444672129781)([18; 18], 0.10691004454512312)([18; 84], 0.016788719751145043)([19; 19], 0.10659353567953948)$

   $([20; 20], 0.09895702894223198)([20; 85], 0.015334943431372141)([21; 21], 0.08569709124073766)([22; 22], 0.06921191456863439)$

   $([23; 23], 0.05167444148670454)([23; 48], 0.00619460237174596)([24; 24], 0.03420388706095644)([25; 25], 0.016187162134860014)$

   After the system decreased the value of ρ from 1 to 0, the above distribution is obtained. Compared to the previous distribution, this distribution is more precise, e.g., it contains the same 15 short "good" intervals, but the sum of their probabilities is higher than in the previous case (0.93 instead of 0.85). This happens because the lower is the value of ρ, the greater number of "good" intervals are identified and left uncoalesced.

8 Conclusions and Future Work

Motivated by the increasing need to analyze complex uncertain multidimensional data (e.g., data from location-based services), in this paper, we introduced the means for managing this data by extending current OLAP/DW technology with *probabilistic* data management.

Specifically, the contributions of this paper were as follows. First, we generalized the notion of *measures* in OLAP data cubes by using *probability distributions* as aggregate values instead of deterministic aggregate values. A probability distribution could capture much more information than a deterministic aggregate value. Specifically, a probability distribution could capture a whole range of possibilities of an aggregate value together with their probabilities, while a deterministic aggregate value could only capture either a summary of the whole range (e.g., an expected value) or one characteristic value from the range (e.g., the maximum value). We represented an aggregate value by a set of integer-bounded *intervals* with each interval assigned its probability. This approach allowed *coalescion* of intervals inside aggregate values in order to gain space efficiency and time efficiency in computations over aggregate values.

Second, we proposed a method for using pre-aggregated aggregate values in order to compute higher-level aggregate values. The method was based on *convolution*, or "summation", of the pre-aggregated probability distributions. In order to make the computations time and space efficient, the method provided mechanisms for *approximate* computation of the probability distributions. Specifically, if several distributions had to be convolved, we "added" the distributions one by one and coalesced intervals in each intermediate result. The precision of the result was controlled by the *maximum number of intervals* in the intermediate results and by the coalescion policies that chose which intervals must remain uncoalesced.

Third, we proposed a method for creating probability distributions from fact data. The method enabled *pre-aggregation* of our generalized measures. This method was an adaptation of the method for using pre-aggregated aggregate values. Specifically, each fact-dimension relation was transformed into a probability distribution and the obtained distributions were convolved.

Finally, we introduced two novel types of probabilistic OLAP queries that operated on the aggregate values represented by the probability distributions and proposed techniques for processing these queries. Specifically, *probability queries* asked for summaries about the distributions (e.g., "For each street in Pullman, WA, what is the probability that the number of cars in the street exceeds 50?"). Then, *aggregation queries* asked for *whole* probability distributions (e.g., "For each street in Pullman, WA, how many cars are in the street?"). Since the queries were processed on coalesced, approximate aggregate values, for both types of queries, we provided *lower bound*, *upper bound*, and *average* approximations of the precise query result.

The concepts presented in the paper were illustrated using a real-world case study from the LBS domain. The work was based on an on-going collaboration with a leading Danish LBS vendor, Euman A/S [8].

Our methods for computing probability distributions and for using the pre-aggregated distributions for further aggregation provided the means for adjusting time efficiency of the computations and precision of the computed distributions. Initial experiments with the methods proved that the means were effective. Specifically, by setting the maximum number of intervals in the resulting count distributions, the system could significantly reduce the time complexity. Moreover, if this lead to unwanted decrease in precision, the precision could be improved by increasing the maximum number of intervals and by other means. Thus, our techniques can be used to strike a good balance between time complexity and precision.

Future work includes generalizing our methods to handle not only probability distributions that capture COUNT aggregate values, but also to handle other aggregate values such as SUM and MIN/MAX. Next, new coalescion policies can be developed to give the system more control over coalescion, which may further increase precision of the computed distributions. Further experiments can be performed. This includes building fact data from real data on moving objects, using probabilistic dimensions (see Section 4.5), using our data selection techniques (see Sections 4.4 and 5.4), and the method extension for LBS (see Section 5.2). Then, it is very important to implement and experiment with our query processing techniques. Finally, it is the most interesting to integrate all the techniques proposed in this paper into existing OLAP systems. This will include incorporating the means for adjusting time efficiency of the computations and precision of the computed distributions into an *automated* adjustment technique. In addition, the techniques from this paper will be extended to better support hyper-dynamic content and future-time queries (e.g., we will develop a method for predicting future changes of pre-aggregated count distributions based on predicted future changes of fact data).

References

[1] An Incomplete Data Cube Project. *http://moab.eecs.wsu.edu/ cdyreson/pub/IncompleteDataCube/*. Current as of August 30, 2005.

[2] D. Barbara, H. Garcia-Molina, and D. Porter. The Management of Probabilistic Data. *Transactions on Knowledge and Data Engineering* 4(5):487–502, 1992.

[3] T. Brinkhoff. A Framework for Generating Network-Based Moving Objects. *Geoinformatica* 6(2):153-180, 2002.

[4] R. Cavallo and M. Pittarelli. The Theory of Probabilistic Databases. In *Proceedings of the 13th International Conference on Very Large Databases (VLDB)*, pp. 71–81, 1987.

[5] N. Dalvi and D. Suciu. Efficient Query Evaluation on Probabilistic Databases. In *Proceedings of the 30th International Conference on Very Large Data Bases (VLDB)*, pp. 864–875, 2004.

[6] M. H. DeGroot and M. J. Schervish. Probability and Statistics. *Addison-Wesley*, 816 pp., 2002.

[7] C. E. Dyreson. Information Retrieval from an Incomplete Data Cube. In *Proceedings of the 22nd International Conference on Very Large Databases*, pp. 532–543, 1996.

[8] Euman A/S. *http://www.euman.com* (in Danish). Current as of August 30, 2005.

[9] E. Gelenbe and G. Hebrail. A Probability Model of Uncertainty in Data Bases. In *Proceedings of the 2nd International Conference on Data Engineering (ICDE)*, pp. 328-333, 1986.

[10] C. Hage, C. S . Jensen, T. B. Pedersen, L. Speičys, and I. Timko. Integrated Data Management for Mobile Services in the Real World. In *Proceedings of the 29th International Conference on Very Large Data Bases (VLDB)*, pp. 1019-1031, 2003.

[11] J. M. Hellerstein, P. J. Haas, and H. J. Wang. Online Aggregation. In *Proceedings of 1997 International Conference on Management of Data (SIGMOD)*, pp. 171–182, 1997.

[12] C. S. Jensen, J. Kolář, T. B. Pedersen, and I. Timko. Nearest Neighbor Queries in Road Networks. In *Proceedings of the 11th International Symposium on Advances in Geographic Information Systems (ACM GIS)*, pp.1–8, 2003.

[13] R. B. Kearfott. Interval Computations: Introduction, Uses, and Resources. *Euromath Bulletin* 2(1):95-112, 1996.

[14] R. Kimball, L. Reeves, M. Ross, and W. Thornthwaite. The Data Warehouse Lifecycle Toolkit. *Wiley*, 800 pp., 1998.

[15] B. R. Moole. A Probabilistic Multidimensional Data Model and Algebra for OLAP in Decision Support Systems. In *Proceedings of IEEE SoutheastCon*, pp. 18–30, 2003.

[16] Oracle Corporation. Oracle 10g Business Intelligence. In *otn.oracle.com/products/bi/content.html*. Current as of August 30, 2005.

[17] T. B. Pedersen, C. S. Jensen, and C. E. Dyreson. Supporting Imprecision in Multidimensional Databases Using Granularities. In *Proceedings of 11th International Conference on Scientific and Statistical Database Management (SSDBM)*, pp. 90–101, 1999.

[18] T. B. Pedersen, C. S. Jensen, and C. E. Dyreson. A Foundation for Capturing and Querying Complex Multidimensional Data. *Information Systems* 26(5):383–423, 2001.

[19] T. B. Pedersen and N. Tryfona. Pre-aggregation in Spatial Data Warehouses. In *Proceedings of the 7th International Symposium on Spatial and Temporal Databases (SSTD)*, pp. 460–478, 2001.

[20] V. Poosala and V. Ganti. Fast Approximate Answers to Aggregate Queries on a Data Cube. In *Proceedings of 11th International Conference on Scientific and Statistical Database Management (SSDBM)*, pp. 24–33, 1999.

[21] V. Poosala, V. Ganti, and Y. E. Ioannidis. Approximate Query Answering using Histograms. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering* 22(4):5–14.

[22] M. S. Puckette. Shannon Entropy and the Central Limit Theorem. Ph.D. Thesis. Department of Mathematics, Harvard University, 1986.

[23] H. M. Regan, S. Ferson, and D. Berleant. Equivalence of Methods for Uncertainty Propagation of Real-Valued Random Variables. *International Journal of Approximate Reasoning* 36:1–30, 2004.

[24] L. Speičys, C. S. Jensen, and A. Kligys. Computational Data Modeling for Network-Constrained Moving Objects. In *Proceedings of the 10th ACM International Symposium on Advances in Geographic Information Systems (ACM GIS)*, pp. 118–125, 2003.

[25] Y. Tao, G. Kollios, J. Considine, F. Li, and D. Papadias. Spatio-Temporal Aggregation Using Sketches. In *Proceedings of the 20th International Conference on Data Engineering (ICDE)*, pp. 214–226, 2004.

[26] I. Timko, C. E. Dyreson, and T. B. Pedersen. Probabilistic Data Modeling and Querying for Location-Based Data Warehouses. In *Proceedings of 17th International Scientific and Statistical Database Management Conference (SSDBM)*, pp. 273–282, 2005.

[27] E. Thomsen, G. Spofford, and D. Chase. Microsoft OLAP Solutions. *Wiley*, 509 pp., 1999.

[28] J. S. Vitter and M. Wang. Approximate Computation of Multidimensional Aggregates of Sparse Data Using Wavelets. In *Proceedings of the 1999 International Conference on Management of Data (SIGMOD)*, pp. 193–204, 1999.

[29] D. Zhang, D. Gunopulos, V. J. Tsotras, and B. Seeger. Temporal and Spatio-Temporal Aggregations over Data Streams Using Multiple Time Granularities. *Information Systems* 28(1–2): 61–84, 2003.

[30] D. Zhang, V. J. Tsotras, and D. Gunopulos. Efficient Aggregation over Objects with Extent. In *Proceedings of the 21st Symposium on Principles of Database Systems (PODS)*, pp. 121–132, 2002.