
Chapter 4. Data Accuracy

This chapter describes our proposal for data accuracy evaluation. We reuse the framework proposed for data freshness evaluation and we propose a data accuracy evaluation algorithm that takes into account the distribution of inaccuracies in source relations. We partition the query result according to data accuracy, labeling each portion with accuracy values and we discuss how such values can be used for enforcing data accuracy.

1. Introduction

The needs of having precise measures of data accuracy become increasingly critical in several fields. Examples are numerous:

- *Information Retrieval*: There may be a great number of web sources providing data to answer a user query and generally a big portion of retrieved data is not relevant for users because of its lacks of accuracy (e.g. hotel lists with incorrect telephone numbers or imprecise prices). The analysis of data accuracy is useful for making a pre-filtering of data or sorting data according to their accuracy.
- *Decision making*: When decision making is based on data extracted from autonomous data sources, external to the organization, a fine knowledge of data quality is necessary in order to associate relative importance to data. In this context, data accuracy should be informed to end-users, as an additional attribute qualifying data. Further strategies, as filtering inaccurate data can also be carried out.
- *Scientific experiments*: Research experiments, especially in the field of life sciences, produce large amounts of data, which are published in databanks and journals. Searches of related experiments are frequently carried out in order to cross results and abstract similar behaviors. Comparison is not trivial and it is worsen by the existence of relatively imprecise data. In this context, the analysis of data accuracy may help reducing the search space in order to retrieve the most accurate experiments.
- *Web-services integration*: When searching for a compatible service among a library of offered services, the selection is usually driven by criteria as response time or service availability. But when the service also provides data, data accuracy may play a critical role, for example, if the service provides yellow pages information.
- *Customer relationship management*: Managing inaccurate data (e.g. wrong customer addresses) may become very expensive, so knowing data accuracy becomes crucial for taking decisions. Furthermore, as many data qualifying customers are obtained from external sources (e.g. address catalogs, yellow pages, census data) with varied quality, data accuracy may be an important factor when choosing among data providers.

All these scenarios motivate the need of data accuracy evaluation methods capable of adapting to different user expectations and different perceptions of data accuracy. As argued in Chapter 2, data accuracy represents a family of quality factors. We recall the three accuracy factors that have been proposed in the literature (see Sub-section 3.1 of Chapter 2 for further details):

- ❑ *Semantic correctness* describes how well data represent states of the real-world. It captures the gap (or the semantic distance) between data represented in the system and real-world data.
- ❑ *Syntactic correctness* expresses the degree to which data is free of syntactic errors such as misspellings and format discordances. It captures the gap (or syntactic distance) between data representation in the system and expected data representation.
- ❑ *Precision* concerns the level of detail of data representation. It captures the gap between the level of detail of data in the system and its expected level of detail.

We consider all these accuracy factors. We use the term *data accuracy* when the discussion concerns all the factors and we refer to *data semantic correctness*, *data syntactic correctness* and *data precision* only when specific discussion is necessary. Concerning accuracy metrics, three types of metrics were described in Sub-section 3.2 of Chapter 2 for any of the accuracy factors:

- *Boolean metric*: It is a Boolean value (1=true, 0=false) that indicates if a data item is accurate or not.
- *Degree metric*: It is a degree that captures the impression or confidence of how accurate is data, commonly represented in the [0-1] range.
- *Value-deviation metric*: It is a numeric value that captures the distance between a system data item and a reference one, generally normalized to the [0-1] range.

We consider all these types of metrics. When an accuracy value must be synthesized from a set of accuracy values, (e.g. for calculating the accuracy of a source relation from the accuracy of individual cells*) we calculate an average of the values. Note that when values are Booleans, the average coincides with a ratio (number of accurate data items in the set divided by the total number of data items in the set). See Sub-section 3.2 of Chapter 2 for further details on aggregation functions.

In this chapter we deal with data accuracy evaluation in data integration systems (DISs). We consider a relational context; specifically we deal with user queries consisting in selections, projections and joins over a set of source relations. We address the problem of evaluating the accuracy of the data conveyed to users in response to their queries and deciding whether users' accuracy expectations can be achieved or not.

We propose partitioning query result in areas (e.g. sets of tuples) having homogeneous accuracy and labeling such areas with their accuracy values. This allows user applications to retrieve only the most accurate data (retrieving the area with the highest accuracy), to filter data not satisfying an accuracy threshold (excluding areas having lower accuracy) or to sort data according to their accuracy (sorting areas by their accuracy). Furthermore, user applications can display first the most accurate area, and if the user wants to see more data (e.g. the result is not complete enough), they can display the following area and so on. This represents a value-added to the conveyed data.

In order to evaluate the accuracy of the data delivered to users and partition query result, we should consider how inaccuracies are distributed in source relations and how they are combined to produce query results. To this end, we partition source relations in areas having homogeneous accuracy. The reason for partitioning is that areas can represent the distribution of inaccuracies in source relations. As argued in [Motro+1998] information sources are rarely of uniform quality, so a unique accuracy value for the whole relation may be a very crude estimation of the accuracy of specific data. Areas are defined as views (selections and projections) over the source relations. In other words, a partition constitutes a set of virtual relations defined by the predicates that characterize the partition.

We reuse the quality evaluation framework proposed for data freshness, modeling the DIS as a quality graph and reducing accuracy evaluation to a problem of value aggregation and propagation through a graph. We present an accuracy evaluation algorithm that takes into account the partitions of source relations and propagates them to query result. We focus on a priori evaluation, i.e. estimating data accuracy before executing user queries. Evaluation results can be used for comparing several query plans in order to choose the one with highest accuracy or combining the K-top plans. Evaluation results can be also used at design time (e.g. for deciding which sources to include in the DIS) and at monitoring time (e.g. for estimating accuracy of test queries). See Section 3.7.1 of Chapter 3 for a description of these usages.

Finally, we discuss the topic of accuracy improvement. We propose to use the partition of query result in order to select the areas that have the best accuracy. Note that we do not select whole relations but the portions that have the best accuracy, which differentiates our approach from the existing source selection approaches.

The following sections describe the approach: Section 2 motivates data accuracy propagation and presents an overview of our approach. Section 3 presents the background knowledge required in this chapter, especially the algorithms that are directly used for accuracy propagation. Section 4 formalizes the evaluation approach and Section 5 suggests some improvement actions. We conclude, in Section 6, by drawing the lessons learned from our experiments.

* The term *cell* refers to an attribute of a tuple.

2. Intuitive approach

In this section we present the intuition of our data accuracy evaluation approach. We consider a DIS in a relational context, providing a global schema that can be queried by users and accessing to a set of source relations that contain data for answering user queries. Our objective is to answer user queries using source data, sorting data in areas (sets of tuples) that have homogeneous accuracy and informing users of the accuracy of such areas. For example, if a user query asks for students' data, we can answer saying '*Student data with 5% of inaccuracies is: ..., student data with 10% of inaccuracies is: ... and so on*'. We describe an *a priori* evaluation strategy, i.e. data accuracy of query results are estimated before executing queries based on estimations of accuracy of source data and on the way of combining it. In this way, only the data that satisfy user's accuracy expectations will be extracted from sources and conveyed in response to the query.

The following example will be used along the chapter for illustrating our evaluation approach. We exemplify the measurement of semantic accuracy with the Boolean metric, but the same discussion can be done for the other accuracy factors and metrics.

Example 4.1. Consider the global schema of a DIS containing two relations:

- S (stid, name, nationality, address, city, telephone, interview, test)
- M (stid, year, mark)

which provide information about students and their annual average marks respectively. Attributes describing students are: stid (the student identification number), name, nationality, address, city, telephone, interview (initial level determined by interviews; taking values 'high', 'medium' or 'low') and test (initial test result; taking values between 0 and 1). Attributes describing marks are: stid (the student identification number), year and mark (taking values in the 0-10 range, where 10 is the maximal mark). The keys of the relations are $\{stid\}$ and $\{stid, year\}$ respectively.

Consider two sources providing information about students and marks:

Source₁:

- $Students$ (stid, name, interview, test, address, telephone) // students living at Montevideo.
- $Marks$ (stid, year, mark) // marks of those students

Source₂:

- $Classing$ (stid, name, nationality, interview, test) // students having a test punctuation superior to 0.8.

Along the chapter, for illustrating some techniques, we will refer to the instances of the *Students* and *Marks* relations of *Source₁*, which are shown Table 4.1 and Table 4.2 respectively. Accuracy values are illustrated coloring the inaccurate cells (Boolean metric). The accuracy values of the *Classing* relation are illustrated in Figure 4.2a also coloring inaccurate cells. Aggregated accuracy values for those relations (obtained as an average of accuracy of cells) are 0.60 (28/60), 0.80 (36/45) and 0.82 (102/125) respectively.

Consider the following queries accessing to S and M :

- $UserQuery_1$ asks for students (stid, name, nationality, interview and test) that obtained 'high' level during interview
- $UserQuery_2$ asks for names and marks of students in year 2005 (keys are also projected)

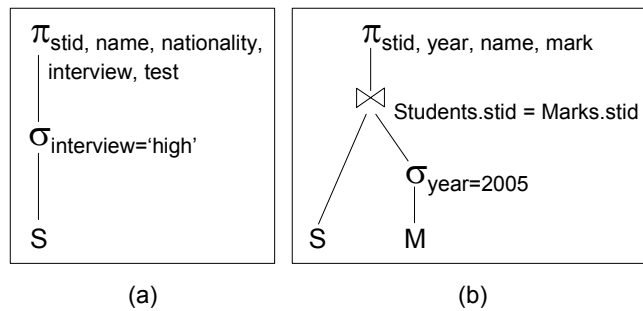
Figure 4.1 illustrates the queries expressed in relational algebra.

Table 4.3 shows a possible answer to $UserQuery_2$, obtained extracting data from the *Students* and *Marks* relations by performing the query $Q_2 = \pi_{stid, year, name, mark}(Students \bowtie_{stid} \sigma_{year='2005'}(Marks))$. Colored cells correspond to inaccuracies. The accuracy of query result is 0.70 (28/40). Analogously, Figure 4.2d shows a possible answer to $UserQuery_1$, obtained extracting data from the *Classing* relation by performing the query $Q_1 = \sigma_{interview='high'}(Classing)$. The accuracy of query result is 0.94 (66/70). □

stid	name	interview	test	address	telephone
21	María Roca	low	1.0	Carrasco	6001104
22	Juan Pérez	medium	.5	Coloniaa 1280/403	9023365
43	Emilio Gutiérrez	high	.8	Irigoitia 384	3364244
56	Gabriel García	low	.5	Propios 2145/101	
57	Laura Torres	medium	.7	Maldonado & Yaro	099628734
58	Raúl González	high	9	Rbla Rca Chile 1280/1102	4112533
101	Carlos Schnider	high	.9701	Copacabana 1210	094432528
102	Miriam Revoir	medium	.7945		9001029
103	A. Benedetti	low	.9146	Charrúa 1284/1	7091232
104	Luis López	high	.8220	Sixtina s/n	

Table 4.1 – *Students* relation

stid	year	mark
21	2005	7
43	2005	10
43	1004	8
56	2004	9
57	2004	3
57	2005	4
58	2005	6
101	2004	9
101	2005	10
102	2004	7
102	2005	10
103	2004	8
103	2005	6
104	2004	10
104	2005	9

Table 4.2 – *Marks* relationFigure 4.1 – User queries: (a) *UserQuery₁*, and (b) *UserQuery₂*

stid	year	name	mark
21	2005	María Roca	7
43	2005	Emilio Gutiérrez	10
57	2005	Laura Torres	4
58	2005	Raúl González	6
101	2005	Carlos Schnider	10
102	2005	Miriam Revoir	10
103	2005	A. Benedetti	6
104	2005	Luis Lopez	9

Table 4.3 – Answer to *UserQuery₂* extracting data from the *Students* and *Marks* relations

Our approach for accuracy evaluation consists in two main phases: (i) partitioning source relations according to accuracy homogeneity in order to represent the distribution of inaccuracies, and (ii) for each user query, partitioning query result based on the partition of source relations.

We partition each source relation in areas (sets of tuples) that are highly homogeneous with respect to their accuracy*. Homogeneity means that any sub-area of a highly homogeneous area would maintain roughly the same accuracy as the initial area. Homogeneity does not mean that all cells in an area have the same accuracy value, but that they can be considered as having the same accuracy value. Areas are defined as views (selections) over the relations. In other words, areas constitute virtual relations defined by the predicates that characterize the partition (the selection predicates) and consequently they can be treated as any relation.

The accuracy of an area can be calculated as the average of the accuracy of its cells (or a sample of cells). The accuracy of cells can be measured using any of the techniques discussed in Chapter 2. When some knowledge about source data is available (e.g. 10% of data about foreign customers is inaccurate), it can be used for estimating accuracy. Such knowledge can be provided by source or domain experts or derived from users' feedback on previously queried data. Note that as areas have homogeneous accuracy, the accuracy of a source relation can also be estimated from the accuracy of its areas, as a weighted sum, where weights are the number of cells in the areas.

* In Section 4, we provide a more complete definition of partitioning, in which areas can be partitioned in sub-areas (sets of attributes) in order to better represent the distribution of inaccuracies. We use a simpler intuition here in order to clearly motivate the proposal.

Example 4.2. Consider the *Classing* relation illustrated in Figure 4.2a (colored cells correspond to inaccuracies). The relation is partitioned in three areas C_1 , C_2 and C_3 , i.e. $Classing = C_1 \cup C_2 \cup C_3$, with 10, 8 and 7 tuples (i.e. 50, 40 and 35 cells) respectively. Accuracy values are aggregated for areas, obtaining the values 0.98 for C_1 , 0.90 for C_2 and 0.50 for C_3 . Figure 4.2b shows the partition, coloring areas with different colors. The accuracy of the relation *Classing* (calculated in previous example as 0.82) can also be calculated from the accuracy of its areas, obtaining $(50 \cdot 0.98 + 40 \cdot 0.90 + 35 \cdot 0.50) / (50 + 40 + 35) = 0.82$. \square

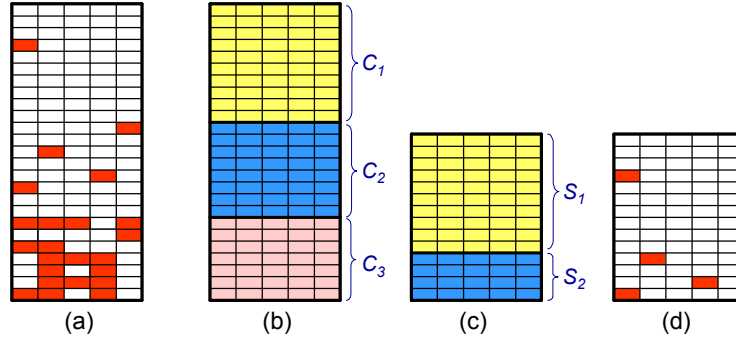


Figure 4.2 – (a) Distribution of inaccuracies in the *Classing* relation (colored cells correspond to inaccuracies), (b) partition of the relation, (c) partition of the answer to *UserQuery₁* extracting data from the *Classing* relation, (d) distribution of inaccuracies in the query answer

In order to build a partition of query result, note that as areas are virtual relations they can be considered as source relations providing data for answering the user query. In other words, given a source relation R partitioned in areas $R_1 \dots R_n$ and an extraction query Q over R , we can define a set of queries $Q_1 \dots Q_n$ with the same semantics as Q , for extracting data from areas $R_1 \dots R_n$, respectively. We call them sub-queries because they extract a portion of the data returned by Q . For example, if $Q = \sigma_P(R)$, we can define $Q_i = \sigma_P(R_i)$, $1 \leq i \leq n$. Furthermore, under certain hypotheses (a set of conditions that will be presented in Sub-section 3.1.4) the union of sub-queries is equivalent to the original query, i.e. $Q \equiv Q_1 \cup \dots \cup Q_n$.

Each sub-query over an area of a source relation returns a set of tuples (eventually empty) that are contained in the result of the original query over the source relation. In other words, the query result is partitioned in areas, each one corresponding to the result of a sub-query. Furthermore, as areas of the source relation have homogeneous accuracy, a subset of their tuples maintains the accuracy, and therefore, the areas of the query result also have homogeneous accuracy.

Example 4.3. Continuing previous example, consider the query $Q_1 = \sigma_{\text{interview}=\text{'high'}}(Classing)$ extracting data from the *Classing* relation in order to solve *UserQuery₁*. The sub-queries S_1 , S_2 and S_3 over C_1 , C_2 and C_3 , defined as $S_i = \sigma_{\text{interview}=\text{'high'}}(C_i)$, $1 \leq i \leq 3$, partition the result of Q_1 , as shown in Figure 4.2c (note that Q_3 does not return any tuple).

The accuracy of S_1 , S_2 and S_3 is estimated as the accuracy of C_1 , C_2 and C_3 , i.e. 0.98, 0.90 and 0.50 respectively, because of the accuracy homogeneity of C_1 , C_2 and C_3 . \square

When several source relations provide the same type of data, the areas of those relations can be combined together. For example, the areas of the *Students* and *Classing* relations can be combined for answering *UserQuery₁*. Furthermore, if the user query joins several relations, sub-queries may join areas that partition several source relations.

Remember that user queries are expressed in terms of the global schema so we need to reformulate (rewrite) them over areas. The problem of rewrite user queries in terms of source relations has been largely studied (it is recalled in Sub-section 3.2). We propose to use query rewriting algorithms for combining areas of source relations instead of combining the whole relations.

Once sub-queries have been generated and having an accuracy estimation for them, we can aggregate an accuracy value for the whole result (as we have done for source relations) by performing a weighted sum of the areas of the result, where weights are the number of cells in each area. To this end, we need to estimate the number of projected attributes and the number of selected tuples in each sub-query. While the former is simple to

obtain (it is structural), the latter needs the estimation of the selectivity of the query. Note that we do not need to know *which* tuples will be returned by each sub-query but *how many* tuples will be returned.

Example 4.4. Continuing previous example, if we estimate that S_1 returns all tuples of C_1 (50 cells), S_2 returns a half of tuples of C_2 (20 cells) and S_3 returns no tuples, the accuracy of Q_1 is calculated as $(50 \cdot 0.98 + 20 \cdot 0.90 + 0 \cdot 0.50) / (50 + 20 + 0) = 0.96$. \square

As areas are not perfectly homogeneous and their accuracy may be an estimation, the accuracy measure obtained for query result is an approximation, not necessarily the exact value that can be computed knowing the precise query result (i.e. with *a posteriori* evaluation). In previous example, we obtained the value 0.96 instead of the exact value $0.94 = 66/70$ obtained from the distribution of inaccuracies shown in Figure 4.2d. But note that the most homogeneous are the areas and the most precise are their accuracy measures, the most precise is the accuracy estimation for the areas of query result. In addition, if the estimation of the size of sub-queries results is good enough, the final accuracy aggregation should be near to the exact accuracy value. Then, the precision of the accuracy estimation relies on the methods used for estimating accuracy of source relations, partitioning them and estimating selectivity of sub-queries.

In summary, our proposal for accuracy evaluation consists in three steps:

1. *Partitioning source relations according to accuracy homogeneity:* This step consists in estimating data accuracy of a sample of each source relation and using accuracy estimations for partitioning the relations, conforming areas of homogeneous accuracy. To this end, we adapt the partitioning algorithm proposed in [Rakov 1998], which is presented in Sub-section 3.1.3. This step is executed once at DIS construction or periodically (but it is not executed for each user query).
2. *Rewriting user queries in terms of partitions:* Each user query is rewritten in terms of the areas that partition source relations. Specifically, a query is expressed as the union of a set of sub-queries, each one accessing to only one area of each source relation. We utilize the Bucket algorithm [Levy+1996] for generating the sub-queries, which is recalled in Sub-section 3.2. The rewriting algorithm can also detect some cases of non-contributive sub-queries, i.e. sub-queries returning no tuples.
3. *Estimating data accuracy of query results:* In this step, we estimate an accuracy value for each sub-query (area in the result) from the accuracy of areas of input relations. This estimation depends on the sub-query operations (selection, projection, join). In addition, the query result is expressed as the union of the sub-queries and an accuracy value is aggregated for query result, as the weighted sum of their accuracy, where weights are the numbers of cells returned by sub-queries. In order to obtain those weights, we estimate the selectivity of operations. Some techniques for selectivity estimation are summarized in Sub-section 3.3.

The quality evaluation framework proposed for data freshness is extended for the evaluation of data accuracy. In this context, quality graphs represent user queries (rewritten in terms of partitions) and are labeled with query properties (such as query selectivity) necessities for accuracy evaluation. The framework is recalled in Sub-section 3.4.

Next section describes the techniques and algorithms that are used in our evaluation approach, and then, Section 4 formalizes the approach.

3. Background

In this section we review the concepts that are used throughout this chapter. We firstly discuss some existing approaches for accuracy evaluation that we adapt to our framework, in particular, a partitioning algorithm, and we recall some properties about fragmentation in the relational model. We also recall the query rewriting principle and we explain the Bucket algorithm. Then, we comment some techniques for selectivity estimation. We end the section recalling the quality evaluation framework presented in Chapter 3.

3.1. Some related approaches for accuracy evaluation

Our approach for accuracy evaluation is based on evaluation techniques proposed in two works: (i) an *a priori* evaluation approach, which assumes uniform distribution of errors [Naumann+1999], and (ii) an approach for a *a posteriori* evaluation, which partitions source relations according to accuracy homogeneity [Rakov 1998]. The following sub-sections describe these approaches.

3.1.1. Techniques for a priori evaluation

A methodology for propagating quality values (including data accuracy) along query operators was proposed in [Naumann+1999]. The approach consists in estimating the accuracy of query results based exclusively in the accuracy of source data. Accuracy of source relations (set granularity) is estimated as the ratio of syntactic correctness (the percentage of cells without errors).

Queries are JSP (join, selection, projection) queries. Authors consider that errors are uniformly distributed in the source relations, so no matter which attributes are projected or which tuples are selected, the accuracy of the source relations is preserved. For the join operation, the accuracy of the joined data is calculated as the product of the accuracy of both input relations. The following example illustrates the approach:

Example 4.5. Consider the query $Q_I = \sigma_{\text{interview}='high'}(\text{Classing})$ introduced in Example 4.1. Selections preserve the accuracy value because of the hypothesis of uniform distribution of errors, so, accuracy of query result is estimated as the accuracy of *Classing*, i.e. 0.82. However, the accuracy of query result is 0.94 (obtained from the distribution of inaccuracies illustrated in Figure 4.2). \square

The weak point of the Naumann's approach is the strong hypothesis on uniform distribution of inaccuracies which is rarely applicable to real data. In most cases, query operations do not preserve accuracy values (e.g. query Q of previous example) and consequently, we do not obtain precise estimations of the accuracy of query results. The main problem is that we do not know where inaccuracies are concentrated (some attributes, some sets of tuples). Additional information describing relation instances is necessary to obtain more precise results. We propose to estimate the distribution of inaccuracies and thus partition source relations.

In next sub-section we discuss an approach for partitioning source relations according to data accuracy, which has been proposed for a posteriori evaluation but can be adapted for a priori evaluation.

3.1.2. Techniques for a posteriori evaluation

An algorithm for partitioning source relations in areas that are highly homogeneous with respect to accuracy was proposed in [Rakov 1998] [Motro+1998]. Areas are defined with views, which may involve selections (with conjunctive conditions) and projections. The accuracy measurement is performed by taking a sample* of each source relation and measuring accuracy of the cells of the sample. Accuracy values for areas (set granularity) are ratios of semantic correctness (percentage of cells that correspond to real-world items). The accuracy values are used for partitioning the sample, using an automatic partitioning algorithm (Algorithm 4.1) that tests different partitioning criteria. Then, the same partition is applied to the whole relation.

User queries are conjunctive queries; query operators are selection, projection and Cartesian product. The relational algebra is extended for operating with the partitions, i.e. operators take as input the relations and their partitions and return a relation and its partition. For example, the partition of a selection or projection is computed intersecting operation result with the partition of the input relation (intersecting projected attributes and selection conditions); accuracy is preserved because of accuracy homogeneity. At the end of the query, a unique accuracy value is calculated for the query result as a weighted sum of the accuracy of the areas (weights are the number of cells in each area). The following example illustrates the approach:

Example 4.6. Consider the query $Q_I = \sigma_{\text{interview}='high'}(\text{Classing})$ presented in Example 4.1 and the partition of the *Classing* relation in the areas C_1 , C_2 and C_3 shown in Figure 4.2b. Areas are defined by certain selection predicates P_1 , P_2 and P_3 , i.e. $C_i = \sigma_{P_i}(R)$, $1 \leq i \leq 3$. Accuracy of areas is 0.98 for C_1 , 0.90 for C_2 and 0.50 for C_3 .

The partition of Q_I results from intersecting the partition of the *Classing* relation with the selection conditions, i.e. $S_i = \sigma_{P_i \wedge \text{interview}='high'}(\text{Classing})$, $1 \leq i \leq 3$. The extension of S_1 , S_2 and S_3 is obtained computing the query, in other words, we know which tuples of each area are selected. Consequently, we know the number of cells of each area that are selected (50, 20 and 0 respectively, as shown in Figure 4.2c). The numbers of selected cells are used as weights for aggregating the accuracy of the result from the accuracy of areas, obtaining: $(50 \cdot 0.98 + 20 \cdot 0.90 + 0 \cdot 0.50) / (50 + 20 + 0) = 0.96$. \square

* Typical size for the sample is 10% of the source relation [Rakov 1998].

The Rakov's approach is based in the knowledge of query result. Specifically, his extended algebra intersects partitions with query results. The difference with our approach is that we do not know which tuples are returned by the query (because we perform a priori evaluation) but we estimate how many tuples may be returned by the query. However, our approach is inspired by the notion of partition of the Rakov's approach. We reuse the partitioning algorithm (which is detailed in Sub-section 3.1.3) and the principle for calculating accuracy of query results as a weighted sum of accuracy of areas.

Another a posteriori approach is proposed in [Laboisse 2005]. They propose measuring and storing accuracy values of source relation cells and storing accuracy values as additional attributes of source relations (called *quality attributes*). Then, when executing user queries quality attributes are also selected, so the accuracy values can be aggregated, obtaining a measure of the accuracy of query result. As this method obtains the precise accuracy value of query result, it will be used for comparing evaluation results.

3.1.3. Partitioning algorithm

This sub-section briefly describes the algorithm for partitioning a relation proposed in [Rakov 1998], whose pseudocode is shown in Algorithm 4.1. It is based on regression and classification trees. It proceeds iteratively, starting at the relation and splitting it in two areas (either horizontally or vertically but not both), then repeating the procedure for each area and so on. At each step, it finds the split that gives maximum gain in homogeneity. The splitting of an area stops when it can provide only marginal improvement in homogeneity (a threshold t is used as stop condition). This indicates that this area has a fairly homogeneous distribution of inaccuracies.

Homogeneity is approximated by Gini indexes [Breiman+1984]. The Gini index $G(v)$ is calculated as $G(v)=2p(1-p)$, where p denotes the proportion of correct cells in a view v . The stop condition calculates the split's reduction of Gini index: $\Delta G = G(v) - \alpha_1 G(v_1) - \alpha_2 G(v_2)$, where $\{v_1, v_2\}$ is a split of v and $\alpha_i = |v_i|/|v|$, $i=1,2$.

As considering all possible splits of a relation is extremely expensive*, Rakov proposes heuristics for reducing the number of splits considered. Recall that there are two sorts of attributes: ordered and categorical. For horizontal splits, if an attribute is ordered and has k distinct values ($a_1 \leq \dots \leq a_k$), they consider the $k-1$ binary conditions $x \leq a_i$ as possible splits. If an attribute is categorical and has l distinct values, they order these values according to the number of inaccurate cells in the tuples having these values, and they treat them as ordered attributes. For vertical splits, if the number of attributes is small, all possible splits can be considered, but if it is large (say, $n > 20$) the same strategy used for categorical attributes can be applied.

```

FUNCTION Partitioning (t: THRESHOLD, R: Relation, G: QualityGraph) RETURNS LIST OF Views
  LIST OF Views V = {};
  QUEUE Q = {R};
  WHILE Q is not empty DO
    Get the next element N of Q;
    Consider all possible splits of N and choose the maximal split s of N;
    IF  $\Delta G(s) * \text{NumberCells}(N) \geq t$ 
      Split N and put the two resulting views in Q;
    ELSE
      Add N to V;
  ENDWHILE;
  RETURN V;
END

```

Algorithm 4.1 – Algorithm for finding a partition of a relation (taken from [Rakov 1998])

This notion of partition is similar to the notion of fragmentation largely studied in distributed databases (see for example [Ozsu+1991]). Next sub-section recalls properties of well-formed fragmentations.

* There are $2^{m-1}-1$ possible horizontal splits and $2^{n-1}-1$ possible vertical splits, being m the number of tuples and n the number of attributes of the relation.

3.1.4. Correctness of fragmentations

Özsu and Valduriez [Ozsu+1991] enunciated three correctness rules that a fragmentation should verify in order to assure database consistency. The rules are:

- *Completeness*: If a relation instance R is decomposed into fragments $R_1 \dots R_n$, each cell that can be found in R can also be found in one or more R_i . This property ensures that the data in a global relation is mapped into fragments without any loss. In the case of horizontal fragmentation the “item” typically refers to a tuple while in the case of vertical fragmentation it refers to an attribute.
- *Reconstruction*: If a relation R is decomposed into fragments $R_1 \dots R_n$, it should be possible to define a relational operator ∇ such that $R = \nabla R_i$, $i = 1..n$. This property ensures that constraints defined on the data in the form of dependencies are preserved. In the case of horizontal fragmentation ∇ is typically the union operation while in the case of vertical fragmentation it is the join operation.
- *Disjointness*: If a relation R is horizontally decomposed into fragments $R_1 \dots R_n$, and cell d_i is in R_j , it is not in any other fragment R_k , $k \neq j$. This criterion ensures that the horizontal fragments are disjoint. If a relation R is vertically decomposed, its primary key attributes are typically repeated in all its fragments. Therefore, in case of vertical fragmentation, disjointness is defined only on the non-primary key attributes of a relation.

Our proposal partitions (fragment) source relations horizontally and vertically. Horizontal partitions must verify the previous rules, i.e. a tuple must belong to one and only one fragment, allowing reconstructing the original relation using the union operator. Vertical partitions must also verify the rules, i.e. a non-key attribute must belong to one and only one fragment. Key attributes must belong to all fragments, allowing reconstructing the original relation using the join operator.

Next sub-section describe another technique reused in this work: query rewriting.

3.2. Query rewriting

In this sub-section we recall the concept of query rewriting and we describe a rewriting algorithm. *Query rewriting* consists in reformulating a user query (expressed in terms of the global schema) into a (possibly) equivalent expression, called *rewriting*, that refers only to the source structures [Calvanese+2001].

In the *local-as-view* (LAV) approach, the global schema is specified independently of the data sources and the mappings between them are established by defining every source relation as a view over the global schema. Expressing sources relations as views over the global schema, the query rewriting problem is similar to the problem of answering queries using views.

Given a query Q over relations E_1, \dots, E_n (the global schema) and a set of views $V = \{V_1, \dots, V_m\}$ (the source relations) over E_1, \dots, E_n , a query Q_r is a rewriting of Q if: (i) it is contained in Q and, (ii) it refers only to V_1, \dots, V_m .

In most works, queries are conjunctive select-project-join queries and are expressed in Datalog-like notation. A query Q has the form:

$$Q(X) \leftarrow R_1(Z_1) \wedge \dots \wedge R_n(Z_n) \wedge C_Q$$

where:

- $R_1 \dots R_n$ are relations; Z_i is a set of variables representing the attributes of R_i *
- C_Q is a conjunction of predicates of the form $u \theta v$ where $\theta \in \{=, <, >, \leq, \geq\}$ and $u, v \in \bigcup_{1 \leq i \leq n} Z_i$
- $X \subseteq \bigcup_{1 \leq i \leq n} Z_i$ is a set of variables representing the attributes projected by Q

Given two queries Q_1 and Q_2 , we say that Q_1 is contained in Q_2 , denoted $Q_1 \subseteq Q_2$, if for all databases the set of tuples returned by Q_1 is included in the set of tuples returned by Q_2 . Query containment has been studied in various works; see for example [Chandra+1977] [Chekuri+1997] [Klug 1988] [van der Meyden 1992].

Several query rewriting algorithms have been proposed; a survey of methods is presented in [Calvanese+2001]. We briefly describe the *Bucket* algorithm [Levy+1996], which will be used later in this chapter.

* Some works allows representing constants in the domain of the attributes. We prefer, for ease of understanding, to represent constants with additional equality conditions. But the two styles are compatible and isomorphic.

The Bucket algorithm aims at computing all the rewritings that are contained in (and not necessarily equivalent to) the original query, by pruning the space of candidate rewritings. It proceeds in two steps:

1. For each atom g in Q , create a bucket that contains the source relations that are contributive for g , i.e. the source relations from which tuples of g can be obtained.
2. Build candidate rewritings (conjunctive queries obtained by combining one source relation from each bucket) and keep only rewritings that are contained in Q .

The first step, whose running-time is polynomial in the number of sources, considerably reduces the number of possibilities considered in the second step. Although containment is intractable in general, its intractability is in the size of the query (which tends to be small) and only occur when queries have multiple occurrences of the same relations; consequently, the complexity of containment is not a problem in practice [Levy+1996].

The following sub-section reviews techniques for selectivity estimation.

3.3. Selectivity estimation

Selectivity estimation techniques are largely used in the field of query optimization; they use statistical information about the data that is stored in the database system to provide estimates to the query optimizer. Histograms are the most common statistical information used in commercial database systems. In this sub-section we show how they are currently used to estimate the selectivity of complex queries.

A histogram on attribute x consists of a set of buckets. Each bucket b_i represents a sub-range r_i of x 's domain, and has associated two values: (i) the frequency f_i , representing the number of tuples t verifying $t.x \in r_i$, and (ii) the distinct value dv_i , representing the number of distinct values of $t.x$ among all the tuples t satisfying $t.x \in r_i$. The main assumption (*uniform spread assumption*) is that each bucket b_i is composed of dv_i equidistant groups of $\delta_i = f_i/dv_i$ tuples each (δ_i is called the density of the bucket) [Bruno+2002]. This assumption suggests a natural interpolation-based procedure to estimate the selectivity of range and join predicates.

To estimate the cardinality of queries with range predicates using a histogram on the range attribute, the frequencies of histogram buckets that are completely or partially covered by the predicate are added (prorating partially covered buckets). Selectivity is obtained dividing query cardinality by relation cardinality.

Example 4.7. Consider the query $Q_3 = \sigma_{R.a < 15}(R)$. Using the histogram on attribute $R.a$ shown in Figure 4.3c the cardinality of Q_3 is estimated adding the frequency of bucket b_1 (which is completely contained) and half of the frequency of bucket b_2 (which is contained at 50% assuming uniform spread), i.e. $60 + 20 \cdot 0.50 = 70$ tuples; selectivity is $70/100 = 0.70$. \square

When queries have multiple range predicates, assuming attribute independence (*independence assumption*) selectivity is estimated as the product of the selectivity of each predicate. For example, given the query $\sigma_{P_1 \wedge P_2}(R)$ where s_i is the selectivity of predicate P_i , $1 \leq i \leq 2$, the selectivity of $P_1 \wedge P_2$ is estimated as $s_1 \cdot s_2$. Multidimensional histograms have been also proposed for the case of non-attribute independence (see for example [Poosala+1997]).

The method to estimate the cardinality of queries with join predicates using histograms on the join attributes consists of three steps [Bruno+2002]. In the first step, both histograms are aligned so that their boundaries agree (splitting some buckets). In the second step, each pair of aligned buckets is analyzed estimating the join size. Assuming that tuples belonging to the bucket with minimal distinct values joins with some tuples of the other bucket (*containment assumption*), the density of the resulting bucket is calculated multiplying the density of input buckets, and frequencies and distinct values are recalculated. In the last step, the cardinality of the join is estimated adding frequencies of buckets. Selectivity is estimated dividing cardinality by the product of cardinalities of input relations.

Example 4.8. Consider the query $Q_4 = R \bowtie_{R.a=S.b} S$. Using the histograms on attributes $R.a$ and $S.b$ shown in Figure 4.3a and Figure 4.3b, the estimation of the cardinality of Q_4 proceeds as follows: Firstly, buckets of both histograms are aligned and frequencies and distinct values are prorated (Figure 4.3c and Figure 4.3d). Then, densities of aligned buckets are calculated multiplying input densities, e.g. $\delta_i'' = 120 (60/2 \cdot 20/5)$, and frequencies and distinct values are aggregated (Figure 4.3e), e.g. $dv_i'' = 2$ (minimum of dv_i and dv_i') and $f_i'' = 240 (\delta_i'' \cdot dv_i'')$. Finally, join cardinality is aggregated, obtaining 440 ($240 + 40 + 160$) tuples. Selectivity is $0.37 = 440 / (100 \cdot 120)$. \square

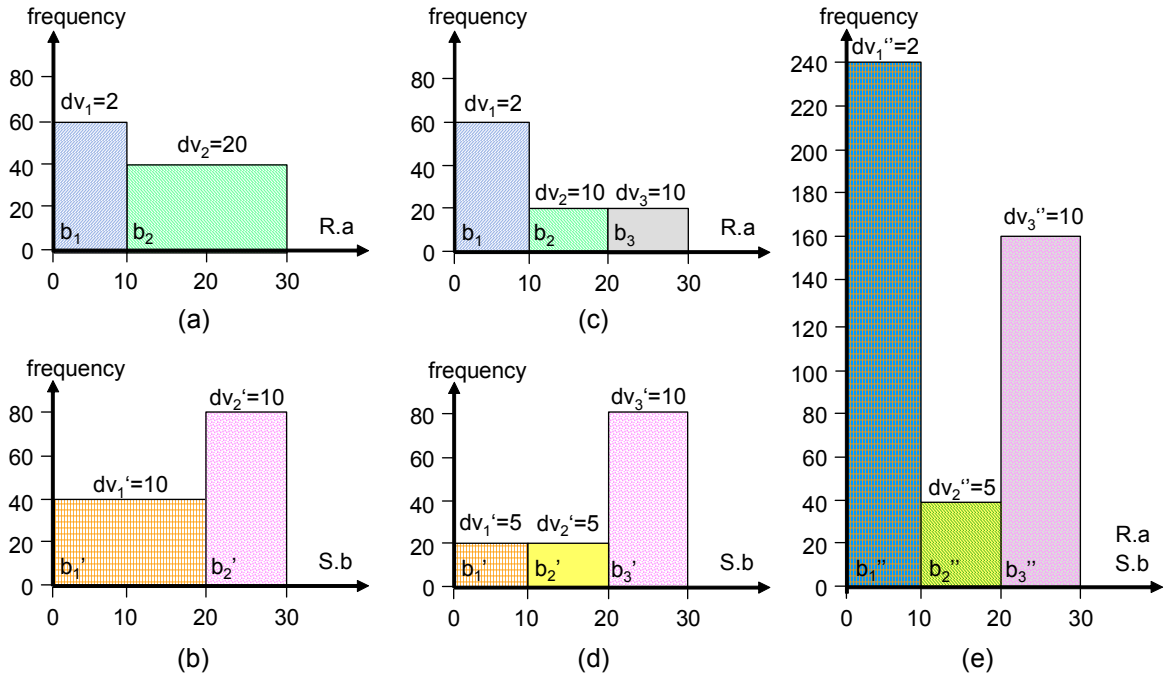


Figure 4.3 - Join selectivity estimation using histograms

When considering arbitrary SPJ queries, we face an additional challenge: cardinality estimation requires propagating statistics through predicates. In other words, we need to build histograms for intermediate results based on the histograms of the base relations.

Example 4.9. Consider the query $Q_5 = \sigma_{R.a < 15}(R \bowtie_{R.a=S.b} S)$ and the histograms shown in Figure 4.3a and Figure 4.3b. The histogram build for the intermediate result $R \bowtie_{R.a=S.b} S$ may be used for estimating selectivity of the selection. \square

As statistics of intermediate results are estimations, their propagation along several query operators may considerably lack of precision. An alternative consists in precalculating statistics of some distinguished query results and using them for calculating selectivities of intermediate results. For example, if statistics for the $R \bowtie_{R.a=S.b} S$ intermediate result are available, they can be used in the estimation of the selectivity of query Q_5 . In [Acharya+1999], authors propose the use of *join synopses* (precomputed samples of a small set of distinguished joins with foreign-key predicates) for building statistics of intermediate results. A more general approach covering JSP queries is presented in [Bruno+2002]. Authors propose to build *SITs* (statistics built on attributes of the result of a query expression) and present methods for selecting the most relevant SITs to compute.

3.4. Quality evaluation framework

This sub-section briefly recall the framework for data quality evaluation presented in Section 2 of Chapter 3. The framework models the DIS processes and properties and evaluates the quality of the data conveyed to the user. The goal of the framework is twofold, firstly, helping in the identification of the DIS properties that should be taken into account for data quality evaluation, and secondly, allowing the easy development of evaluation algorithms that consider such properties.

The framework consists of: (i) a set of data sources, (ii) a set of data targets, (iii) a set of quality graphs representing several DISs processes, (iv) a set of properties describing DIS features and quality measures, and (v) a set of quality evaluation algorithms.

A DIS is modeled as a workflow process in which the workflow activities perform the different tasks that extract, integrate and convey data to end-users. Each workflow activity takes data from sources or other activities and produces result data that can be used as input for other activities. Then, data traverses a path from sources to targets where it is transformed and processed according to the system logics. A *quality graph* is a

graph that has the same workflow structure as the DIS and is adorned with property values useful for quality evaluation. The nodes are of three types: (i) *activity nodes* representing the major tasks of a DIS, (ii) *source nodes* representing data sources accessed by the DIS, and (iii) *target nodes* representing data targets fed by the DIS. There are two types of edges: (i) *control edges* expressing the control flow dependencies between activities, and (ii) *data edges* representing data flow from sources to activities, from activities to targets and between activities. In the DISs considered in this chapter, control flow is induced by data flow, i.e. there is a control flow edge between two activities if and only if there is a data flow edge between them. Nodes and edges of quality graphs are adorned with property labels of the form *property = value*. Properties can be of two types: (i) *features*, indicating some characteristic of the DIS (costs, delays, policies, strategies, constraints, etc.), or (ii) *measures*, indicating a quality value corresponding to a quality factor. Figure 4.4 illustrates a quality graph.

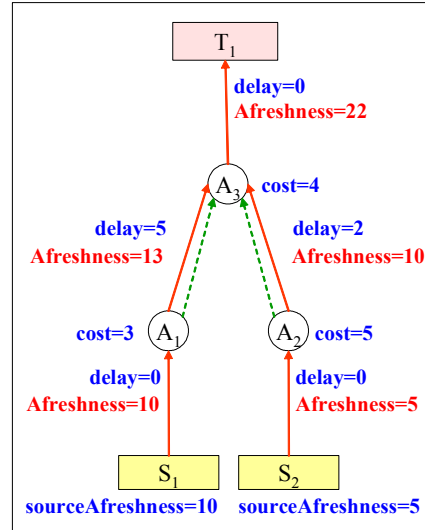


Figure 4.4 – Quality graph

The quality evaluation is performed by evaluation algorithms. As a quality graph describes the DIS integration process and its properties, it contains the input information needed by evaluation algorithms. Evaluation algorithms take as input a quality graph, calculate the quality values corresponding to a quality factor and return a quality graph with an additional property (corresponding to the evaluated quality factor). Evaluation algorithms may traverse the graph, node by node, operating with property values; this mechanism is called *quality propagation*. Concerning code, evaluation algorithms have the following signature:

FUNCTION AlgorithmName (G: QualityGraph) **RETURNS** QualityGraph

The implementation of evaluation algorithms may vary according to the quality factor and the concrete application scenario. The framework does not constrain the way the algorithms can be implemented.

Next section formalizes our approach for data accuracy evaluation. The techniques and algorithms described in this section are adapted and used during accuracy evaluation.

4. Formal approach

In this section we formalize our approach for accuracy evaluation. Accuracy can be measured using any of the techniques discussed in Chapter 2. Our approach is independent of the accuracy metric used but of course the accuracy measures depend on it. We consider a DIS in a relational context, i.e. it integrates data of relational sources and allows users to pose JSP (join, selection, projection) queries over a relational global schema. User queries are reformulated (rewritten) in terms of the source relations, obtaining a set of query rewritings that provide data for answering the user query.

We address the problem of estimating the accuracy of the data conveyed to users in response to a query. We organize data in areas of homogeneous accuracy in order to inform users (or user applications) about the distribution of inaccuracies. To this end, source relations are partitioned in areas having homogeneous accuracy.

Areas are virtual relations (views) defined according to the partitioning criteria (selection conditions and projection attributes). User queries are rewritten over these areas (instead of over whole relations).

We propose an *a priori* evaluation strategy, i.e. data accuracy is estimated before executing the rewriting queries, based exclusively on the accuracy of the areas of the source relations and on the operations (selection, projection, join) that define the rewriting.

The problem can be stated as follows:

Given:

- a set of source relations partitioned in areas of homogeneous accuracy, and
- a user query over the global schema,

Obtain:

- query rewritings (over areas of source relations) that answer the query, and
- estimations of the accuracy of their data

Our proposal for accuracy evaluation consists in three steps:

1. *Partitioning source relations according to accuracy homogeneity*: This step consists in estimating data accuracy of a sample of each source relation and using accuracy estimations for partitioning the relations. This step is executed in a preparation phase, at DIS construction or periodically, but separated from the query evaluation phase. It is described in Sub-section 4.1.
2. *Rewriting user queries in terms of partitions*: Each user query is rewritten in terms of the areas that partition source relations. This step consists in generating a set of query rewritings over these areas. The result to the user query consists of the union of the generated rewritings. This step is explained in Sub-section 4.2.
3. *Estimating data accuracy of query results*: In this step, we estimate the accuracy of the data returned by each rewriting, based on the accuracy of areas of source relations, and we aggregate an accuracy value for query result. The estimation is described in Sub-sections 4.3 and an accuracy evaluation algorithm is proposed in Sub-section 4.4.

In order to enforce data accuracy, a simple improvement action consists in discarding rewritings (or directly areas of source relations) that have low accuracy. Note that areas can be discarded in an early phase (during the generation of the rewritings) so our proposal correspond to a *selective rewriting* strategy. This point is discussed in Sub-section 5.

The following sub-sections describe each step.

4.1. Partitioning of source relations according to accuracy homogeneity

Inspired by the partitions proposed by Rakov [Rakov 1998], we partition each source relation according to data accuracy. The idea behind partitioning is to manipulate portions of the relation that have homogeneous accuracy, i.e. if we partition again, in any manner, the accuracy estimation will not considerably change. In this sub-section we define partitions and we discuss how a convenient partition can be obtained.

For easy manipulation of partitions (which will be explained in next sub-section), we firstly partition the relation in *areas* according to selection predicates (horizontal partition) and then, we partition each area in *sub-areas* according to sub-sets of attributes (vertical partition). Partitions must verify the three correctness rules discussed in Sub-section 3.1.2, i.e. a tuple must belong to one and only one area, a non-key attribute must belong to one and only one sub-area and key attributes must belong to all sub-areas.

In order to treat all attributes (key and non-key attributes) in the same manner, i.e. projecting them in only one sub-area, we duplicate key attributes, generating a new key for the relation (that composed of the new attributes). This strategy was introduced by Rakov [Rakov 1998] with the name of *key expansion*.

Definition 4.1 (key expansion). Given a relation $R(A_1, \dots, A_m, \dots, A_n)$, $m \leq n$, where A_1, \dots, A_m constitute a key of R , a *key expansion* of R , denoted \bar{R} , is a relation obtained replicating the key attributes: $\bar{R}(K_1, \dots, K_m, A_1, \dots, A_m, \dots, A_n)$. The replicated attributes (K_1, \dots, K_m) conform a key of \bar{R} , which is called *expanded key*. \square

In vertical partitions, the expanded key is projected in all sub-areas and each attribute of the original relation is projected in only one sub-area. We formalize the concepts of partition, area and sub-area as follows:

Definition 4.2 (horizontal partition). Given a relation R , its key expansion \bar{R} and a set of conjunctive predicates $\{P_1, \dots, P_m\}$ over R , complete and disjoint (i.e. each tuple of R verifies one and only one predicate), a *horizontal partition* of R , denoted $\mathcal{HP}_{P_1 \dots P_m}(R)$, is a set of selection views $\{R_1, \dots, R_m\}$, called *areas*, obtained applying the predicates P_1, \dots, P_m to \bar{R} .

$$\mathcal{HP}_{P_1 \dots P_m}(R) = \{R_1, \dots, R_m\} = \{\sigma_{P_1}(\bar{R}), \dots, \sigma_{P_m}(\bar{R})\}$$

We denote an area by a 4-uple $\langle \text{Name}, \text{Predicate}, \text{NumberTuples}, \text{KeyAccuracy} \rangle$ where *Name* is a name that identifies the area with respect to the relation, *Predicate* is the conjunctive predicate that defines the area (noted between square brackets), *NumberTuples* is the estimated number of tuples verifying the predicate and *KeyAccuracy* is the estimated accuracy of the key attributes of R . \square

Definition 4.3 (vertical partition). Given an area R_i of relation R and n disjoint subsets of attributes of R ($S_1 \dots S_n$), a *vertical partition* of R_i , denoted $\mathcal{VP}_{S_1 \dots S_n}(R_i)$, is a set of projection views $\{R_{i1}, \dots, R_{in}\}$, called *sub-areas*, obtained projecting the subsets of attributes to R_i . Each attribute of R is projected in one and only one sub-area. The expanded key K of \bar{R} is projected in all sub-areas.

$$\mathcal{VP}_{S_1 \dots S_n}(R_i) = \{R_{i1}, \dots, R_{in}\} = \{\pi_{K, S_1}(R_i), \dots, \pi_{K, S_n}(R_i)\}$$

We denote a sub-area by a 3-uple $\langle \text{Name}, \text{Attributes}, \text{Accuracy} \rangle$ where *Name* is a name that identifies the sub-area with respect to the relation, *Attributes* is the subset of attributes that defines the sub-area and *Accuracy* is the estimated accuracy of the sub-area. \square

Areas and sub-areas are virtual relations (views) defined by the partitioning criteria (selection conditions and projection attributes), i.e. source relations are not physically fragmented and stored as a set of independent fragments. Source relations are kept unchanged in data sources and metadata describing areas and sub-areas (which was specified in Definition 4.2 and Definition 4.3) is stored at the DIS.

The following example illustrates the definition of partitions:

Example 4.10. Consider the *Students*(stdid, name, interview, test, address, telephone) and *Marks*(stdid, year, mark) relations presented in Example 4.1. The key expansion of relations are: *Students*(Kstdid, stdid, name, interview, test, address, telephone) and *Notes*(Kstdid, Kyear, stdid, year, mark), where $\{Kstdid\}$ and $\{Kstdid, Kyear\}$ are the expanded keys.

In this step the relations are partitioned in areas and sub-areas and metadata (number of tuples, accuracy values) is estimated. A possible partition of the *Students* relation can be:

- Area S_1 ; [stdid < 101]; 6 tuples; key accuracy=0.50
 - Sub-area S_{11} ; {stdid, name, interview, test}; accuracy=0.50
 - Sub-area S_{12} ; {address, telephone}; accuracy=0.25
- Area S_2 ; [stdid ≥ 101]; 4 tuples; key accuracy=1.00
 - Sub-area S_{21} ; {stdid}; accuracy=1.00
 - Sub-area S_{22} ; {name, interview, test, address, telephone}; accuracy=0.87

Analogously, a possible partition of the *Students* relation can be:

- Area M_1 ; [year < 2004]; 1 tuple; key accuracy=0.00
 - Sub-area M_{11} ; {stdid, year, mark}; accuracy=0.00
- Area M_2 ; [year ≥ 2004 ∧ stdid < 101]; 6 tuples; key accuracy=0.67
 - Sub-area M_{21} ; {stdid, year, mark}; accuracy=0.67
- Area M_3 ; [year ≥ 2004 ∧ stdid ≥ 101]; 8 tuples; key accuracy=1.00
 - Sub-area M_{31} ; {stdid, year, mark}; accuracy=1.00 \square

After partitioning a relation (horizontally and vertically), each cell of the relation belongs to a unique sub-area of a unique area, so we can represent partitions assigning different colors or patterns to each sub-area. Table 4.4 illustrates the partition of the *Students* relation introduced in previous example. As the expanded key belongs to all sub-areas, it is not colored and can be omitted in the graphical representation.

Kstid	stid	name	interview	test	address	telephone
21	21	María Roca	low	1.0	Carrasco	6001104
22	22	Juan Pérez	medium	.5	Coloniaa 1280/403	9023365
43	43	Emilio Gutiérrez	high	.8	Irigoitia 384	3364244
56	56	Gabriel García	low	.5	Propios 2145/101	
57	57	Laura Torres	medium	.7	Maldonado & Yaro	099628734
58	58	Raúl González	high	9	Rbla Rca Chile 1280/1102	4112533
101	101	Carlos Schnider	high	.9701	Copacabana 1210	094432528
102	102	Miriam Revoir	medium	.7945		9001029
103	103	A. Benedetti	low	.9146	Charrúa 1284/1	7091232
104	104	Luis López	high	.8220	Sixtina s/n	

Table 4.4 – Graphical representation of areas and sub-areas

In order to partition source relations, we can use the Rakov's algorithm presented in Sub-section 3.1.3. But remember that the algorithm alternates horizontal and vertical splits and consequently, the obtained partition may not correspond a set of areas decomposed in sub-areas. For example, the partition of Figure 4.5a can be returned by the Rakov's algorithm. We have to restructure the partition according to Definition 4.2 and Definition 4.3.

Given any hybrid partition of a relation R consisting in a set of fragments $\{F_1, \dots, F_k\}$ verifying correctness rules, we can obtain another partition of R that follows Definition 4.2 and Definition 4.3 by further partitioning some of the fragments. In other words, we can obtain a set of sub-areas $\{S_{11}, \dots, S_{1m_1}, \dots, S_{n1}, \dots, S_{nm_n}\}$ in which the sub-sets $\{S_{i1}, \dots, S_{i1_i}\}$ vertically partition a certain area A_i , $1 \leq i \leq n$, and the set of areas $\{A_1, \dots, A_n\}$ horizontally partition R . We proceed as follows:

1. We obtain the set of selection predicates $P = \{P_1, \dots, P_r\}$ that define the fragments F_1, \dots, F_k , $r \leq k$. Note that r can be smaller than k since several fragments can have the same selection predicate.
2. While there exist in P two predicates P_i and P_j with a common sub-expression, i.e. $P_i \cap P_j \neq \emptyset$, we substitute P_i and P_j by $P_i \cap P_j$, $P_i - P_j$ and $P_j - P_i$. Since the predicates are conjunctions of simple inequalities on the attributes of R , this step will finish. We obtain a set of disjoint predicates $P' = \{P'_1, \dots, P'_n\}$.
3. We define a set of areas $\{A_1, \dots, A_n\}$ one for each predicate in P' . Note that $\{A_1, \dots, A_n\}$ conforms a horizontal partition of R , i.e. it is disjoint because the predicates in P' are disjoint and it is complete because $\{F_1, \dots, F_k\}$ was complete and we do not loose sub-expressions of predicates.
4. We intersect each area A_i with the fragments F_1, \dots, F_k , obtaining a set of sub-areas $\{S_{i1}, \dots, S_{i1_i}\}$. Note that $\{S_{i1}, \dots, S_{i1_i}\}$ conforms a vertical partition of A_i , i.e. it is complete and disjoint because $\{F_1, \dots, F_k\}$ was complete and disjoint.

Example 4.11. Consider the hybrid partition of relation R shown in Figure 4.5a. Let P_i be the selection predicate of fragment F_i , $1 \leq i \leq 7$. We define $X = P_3 \cap P_4$. Observe that $P_5 = P_6 = P_7$, $P_3 = P_2 \cup X$ and $P_4 = X \cup P_5$. Then, the set of disjoint predicates obtained in step 2 is $P' = \{P_1, P_2, X, P_5\}$, which define the areas $\{A_1, A_2, A_3, A_4\}$ shown in Figure 4.5b. Intersecting areas with the original fragments, we obtain the sub-areas $\{S_{11}, S_{21}, S_{22}, S_{31}, S_{32}, S_{41}, S_{42}, S_{43}, S_{44}\}$ shown in Figure 4.5b.

Then, any hybrid partition of a relation (satisfying correctness rules) can be restructured in areas and sub-areas. In particular, the partition produced by the Rakov's algorithm can be expressed in this way.

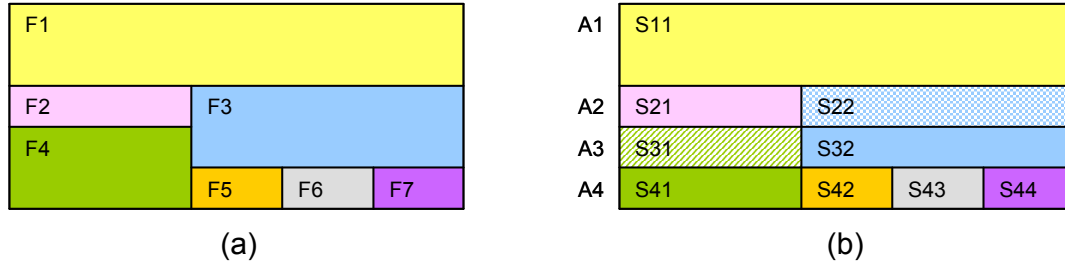


Figure 4.5 – Restructure of an arbitrary partition in areas and sub-areas: (a) partition before restructure and (b) partition after restructure

In order to partition source relations, we propose to use the Rakov's algorithm, however, areas and sub-areas can be also defined manually, using knowledge about source relations obtained from source administrators, third-party consultants, experts or users' feedback.

The Rakov's algorithm computes the accuracy of an area as the proportion of semantically correct cells. Such accuracy value (p) is the parameter for calculating the Gini indexes $G(v)=2p(1-p)$, which are used to compute accuracy homogeneity and consequently to choose the split that represents maximal gain in homogeneity (see Sub-section 3.1.3 for details). In other words, the partitioning algorithm can be parameterized in order to calculate other accuracy metric, providing a function for computing the accuracy of areas.

The partition obtained applying the Rakov's algorithm is restructured as previously explained. Metadata describing areas and sub-areas is also obtained from the partition, as follows:

- Areas are sequentially named as well as their sub-areas.
- The predicates defining areas and the sub-sets of attributes defining sub-areas are taken from the partition.
- The accuracy of a sub-area is estimated as the average of the accuracy of the cells of the sub-area. This estimation is also calculated by Rakov's algorithm.
- Accuracy of keys is estimated as the average of the accuracy of attributes that conform the key (accuracy of attributes is the accuracy of the sub-areas they belong because of accuracy homogeneity). Note that when all key attributes are projected in a unique sub-area, the accuracy of the key coincides with the accuracy of the sub-area.
- Finally, remember that the Rakov's algorithm partitions a sample of the source relation, so the number of tuples of an area is extrapolated from the number of tuples of the area in the sample, i.e. multiplying it by the ratio between relation size and sample size.

Partitioning is performed once (at DIS construction) or periodically, but it is separated from the evaluation of accuracy of user queries. Next sub-section describes how queries are rewritten in terms of partitions and its impact in accuracy evaluation.

4.2. Rewriting of user queries in terms of partitions

We propose to rewrite user queries in terms of areas of source relations. At the end of the sub-section we consider the alternative of rewriting user queries in terms of sub-areas and we explain why we discarded this option.

In order to express user queries in terms of the areas of source relations, we propose using classical query rewriting algorithms, as the Bucket algorithm [Levy+1996]. To this end, areas must be expressed as views of the global schema, following the local-as-view (LAV) approach. This sub-section explains the rewriting principle.

In the local-as-view (LAV) approach source relations are expressed as views of the global schema. As each area is a view over a source relation, it can be easily expressed as a view of the global schema by unfolding the view over the source relation, i.e. substituting the source relation with its definition in terms of the global schema.

The expression of areas in terms of the global schema is independent of user queries, i.e. it is done after partitioning source relations, but it is separated from the evaluation of accuracy of user queries.

Example 4.12. Continuing Example 4.10, the *Students* and *Marks* relations are expressed in terms of the global schema (relations *S* and *M*) as follows:

- $\text{Students}(\text{id}, \text{na}, \text{in}, \text{te}, \text{a}, \text{tp}) \leftarrow S(\text{id}, \text{na}, \text{nt}, \text{a}, \text{c}, \text{tp}, \text{in}, \text{te}) \wedge \text{c} = \text{'Montevideo'}$
- $\text{Marks}(\text{id}, \text{y}, \text{m}) \leftarrow M(\text{id}, \text{y}, \text{m})$

Remember the definition of their areas (expressed in Datalog-like notation):

- $S_1(\text{id}, \text{na}, \text{in}, \text{te}, \text{a}, \text{tp}) \leftarrow \text{Students}(\text{id}, \text{na}, \text{in}, \text{te}, \text{a}, \text{tp}) \wedge \text{id} < 101$
- $S_2(\text{id}, \text{na}, \text{in}, \text{te}, \text{a}, \text{tp}) \leftarrow \text{Students}(\text{id}, \text{na}, \text{in}, \text{te}, \text{a}, \text{tp}) \wedge \text{id} \geq 101$
- $M_1(\text{id}, \text{y}, \text{m}) \leftarrow \text{Marks}(\text{id}, \text{y}, \text{m}) \wedge \text{y} < 2004$
- $M_2(\text{id}, \text{y}, \text{m}) \leftarrow \text{Marks}(\text{id}, \text{y}, \text{m}) \wedge \text{y} \geq 2004 \wedge \text{id} < 101$
- $M_3(\text{id}, \text{y}, \text{m}) \leftarrow \text{Marks}(\text{id}, \text{y}, \text{m}) \wedge \text{y} \geq 2004 \wedge \text{id} \geq 101$

Substituting *Students* and *Marks* by their definitions we obtain the expression of areas in terms of the global schema:

- $S_1(\text{id}, \text{na}, \text{in}, \text{te}, \text{a}, \text{tp}) \leftarrow S(\text{id}, \text{na}, \text{nt}, \text{a}, \text{c}, \text{tp}, \text{in}, \text{te}) \wedge \text{c} = \text{'Montevideo'} \wedge \text{id} < 101$
- $S_2(\text{id}, \text{na}, \text{in}, \text{te}, \text{a}, \text{tp}) \leftarrow S(\text{id}, \text{na}, \text{nt}, \text{a}, \text{c}, \text{tp}, \text{in}, \text{te}) \wedge \text{c} = \text{'Montevideo'} \wedge \text{id} \geq 101$
- $M_1(\text{id}, \text{y}, \text{m}) \leftarrow M(\text{id}, \text{y}, \text{m}) \wedge \text{y} < 2004$
- $M_2(\text{id}, \text{y}, \text{m}) \leftarrow M(\text{id}, \text{y}, \text{m}) \wedge \text{y} \geq 2004 \wedge \text{id} < 101$
- $M_3(\text{id}, \text{y}, \text{m}) \leftarrow M(\text{id}, \text{y}, \text{m}) \wedge \text{y} \geq 2004 \wedge \text{id} \geq 101 \quad \square$

In order to rewrite user queries in terms of areas, the *Bucket* algorithm first create buckets, comparing query predicates with area predicates and then generates the candidate rewritings, checking query containment. The Bucket algorithm was described in Sub-section 3.2. The following example shows its use:

Example 4.13. Consider *UserQuery₂* presented in Example 4.1, expressed in Datalog-like notation:

- $\text{UserQuery}_2(\text{id}, \text{y}, \text{n}, \text{m}) \leftarrow S(\text{id}, \text{n}, \text{nt}, \text{a}, \text{c}, \text{tp}, \text{in}, \text{te}) \wedge M(\text{id}, \text{y}, \text{m}) \wedge \text{y} = 2005$

In order to rewrite *UserQuery₂* in terms of S_1 , S_2 , M_1 , M_2 and M_3 discussed in previous example, the following buckets are created: $\text{Buckets}(S) = \{S_1, S_2\}$ and $\text{Buckets}(M) = \{M_2, M_3\}$. The area M_1 is not included in the second bucket because it is contradictory to the query predicate ('year < 2004' and 'year = 2005').

The following rewritings are generated, taking an area of each bucket:

- $QR_1(\text{id}, \text{y}, \text{n}, \text{m}) \leftarrow S_1(\text{id}, \text{n}, \text{in}, \text{te}, \text{a}, \text{tp}) \wedge M_2(\text{id}, \text{y}, \text{m}) \wedge \text{y} = 2005$
- $QR_2(\text{id}, \text{y}, \text{n}, \text{m}) \leftarrow S_2(\text{id}, \text{n}, \text{in}, \text{te}, \text{a}, \text{tp}) \wedge M_3(\text{id}, \text{y}, \text{m}) \wedge \text{y} = 2005$

Area S_1 is not combined with area M_3 because they are contradictory ('id < 101' and 'id ≥ 101'); analogously, area S_2 is not combined with area M_2 . So, $\text{UserQuery}_2 \supseteq QR_1 \cup QR_2$.

We only considered the areas of the *Students* and *Marks* relations for reducing the size of the example, but in order to rewrite *UserQuery₂* the areas of the *Classing* relation (areas C_1 , C_2 and C_3 discussed in Example 4.2) must be also considered for being added to $\text{Bucket}(S)$. Consequently, new query rewritings may result from combining C_1 , C_2 and C_3 with M_2 and M_3 . \square

The number of rewriting grows polynomially with the number of areas. As proved in [Levy+1996], the intractability of rewriting algorithms is not in the number of relations but in the size of the query (which tends to be small) and only occurs when queries have multiple occurrences of the same relations.

We consider now the alternative of rewriting user queries in terms of sub-areas, explaining why we discarded it. Areas keep whole tuples of source relations. Therefore, when rewriting queries over areas, the rewriting algorithm joins whole tuples of sources relations (which is analogous to rewrite queries over source relations). However, sub-areas break tuples because they project some attributes of a relation. Firstly, we remark that there are extensions to the Bucket algorithm that consider the join of several relations of the same bucket in order to provide all required attributes, so query rewriting over sub-areas is possible. However, joining few attributes of a lot of relations seriously increase the risk of introducing semantic inconsistencies, i.e. building tuples that have no sense in real world. For example, we can return the name of a student and the telephone of another student, both having the same student id in different source databases. This risk is inherent to DIS but it extremely increases when we increase the number of joins (i.e. when we break tuples). In addition, rewriting algorithms avoid breaking tuples when possible for reducing this risk.

For this reason, we choose to rewrite user queries over areas and logically maintain sub-area metadata for calculating data accuracy. This is also the reason for partitioning source relations horizontally and then vertically instead of managing arbitrary hybrid partitions as in [Rakov+1998]. In next sub-section we evaluate data accuracy for each rewriting.

4.3. Estimation of data accuracy of query results

In order to estimate accuracy of areas and aggregate an accuracy value for the query result, we proceed in three sub-steps: (i) determining areas and sub-areas of the rewriting, (ii) estimating accuracy and key accuracy of the rewriting, and (iii) estimating the number of tuples of each area for performing the aggregation. Next sub-sections describe each sub-step.

4.3.1. Determination of the areas and sub-areas of a rewriting

Each rewriting is defined as the join of several areas, each area containing some (eventually one) sub-areas. The result of the join is one area, which satisfies the selection predicates of all input areas, and contains the union of all sub-areas having some of the projected attributes. Next example illustrates this.

Example 4.14. Figure 4.6 illustrates a query rewriting that joins three areas (A_1 , A_2 and A_3), each one having several sub-areas (represented with different colors and patterns). The rewriting consists of an area (QR) with the union of the sub-areas that contain some of the projected attributes. Sub-area A_{12} does not belong to QR because none of its attributes are projected in the rewriting. \square

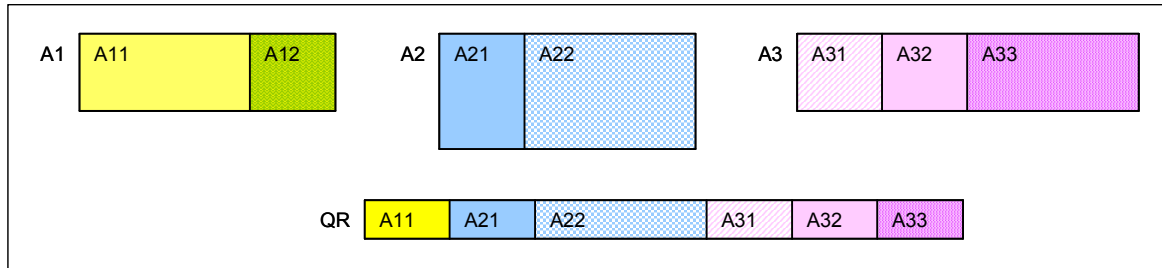


Figure 4.6 – Joining several areas

Each tuple returned by the rewriting will satisfy the predicates of all areas. So we build a unique area for the rewriting, as a conjunction of the predicates of the areas and the predicates of the rewriting. Note that predicates cannot be contradictory because the *Bucket* algorithm only returns contributive rewritings. Predicates that are less restrictive than other ones are not listed (e.g. ‘year = 2005’ is more restrictive than ‘year ≥ 2004’).

The sub-areas of the rewriting are the union of the sub-areas of the input areas, intersecting attributes with the rewriting attributes. If the rewriting does not project any of the attributes of a sub-area, the sub-area is eliminated. The sub-areas of the rewriting conform a vertical partition of its unique area. The satisfaction of the completeness property is straightforward because all attributes of the rewriting belong to some sub-area of the inputs areas. Regarding the disjointness property, a problem can arise with natural join because, join attributes belong to two input sub-areas (which is necessary for performing the join) but are projected only once in the rewriting. The following example illustrates this case:

Example 4.15. Consider the rewriting QR_I of Example 4.13:

$$QR_I(id, y, n, m) \leftarrow S_1(id, n, in, te, a, tp) \wedge M_2(id, y, m) \wedge y=2005$$

The *id* variable is used in areas S_I and M_2 , conforming a natural join. The *id* variable represents an attribute of S_I (which is projected in a sub-area of S_I) and an attribute of M_2 (which is projected in a sub-area of M_2). But it also represents an attribute of the rewriting, which must be projected in a unique sub-area. \square

Each natural join attribute must be projected in a unique sub-area of the rewriting in order to satisfy the *disjointness* property required for partitions. To this end, we create a new sub-area for the attribute, whose accuracy value will be calculated as an average of the accuracy of both sub-areas, as will be explained in next sub-section. After calculating accuracy, sub-areas having the same accuracy value will be fusioned.

Example 4.16. Continuing Example 4.13, areas and sub-areas for QR_1 , and QR_2 are calculated as follows: One area is created for each rewriting, QR_1 and QR_2 respectively, labeled with the input areas and the rewriting predicates. Rewriting QR_1 joins areas S_1 and M_2 , whose sub-areas are S_{11} , S_{12} and M_{21} . Sub-areas QR_{11} and QR_{12} result from intersecting attributes of sub-areas S_{11} and M_{21} with rewriting attributes; the join attribute is separated in sub-area QR_{13} . No attribute of sub-area S_{12} is projected. Sub-areas of QR_2 are calculated analogously. We obtain:

- Area QR_1 {stid < 101 \wedge year = 2005}; ? tuples; key accuracy = ?; inputs: S_1 and M_2
 - Sub-area QR_{11} {name}; accuracy = ?; input: S_{11}
 - Sub-area QR_{12} {year, mark}; accuracy = ?; input: M_{21}
 - Sub-area QR_{13} {stid}; accuracy = ?; inputs: S_{11} and M_{21}
- Area QR_2 {stid \geq 101 \wedge year = 2005}; ? tuples; key accuracy = ?; inputs: S_2 and M_3
 - Sub-area QR_{21} {name}; accuracy = ?; input: S_{22}
 - Sub-area QR_{22} {year, mark}; accuracy = ?; input: M_{31}
 - Sub-area QR_{23} {stid}; accuracy = ?; inputs: S_{21} and M_{31}

Input areas or sub-areas where also registered for understanding purposes. \square

Next sub-section explains how to calculate accuracy values and Sub-section 4.3.3 discusses the estimation of the number of tuples.

4.3.2. Accuracy calculation

If partitions are quite well defined, they should reasonably preserve the accuracy of sub-areas, being insensible to projection attributes and selection predicates, because of accuracy homogeneity. Regarding joins, as a tuple is build from two input tuples, the accuracy of resulting sub-areas may depend on both input areas. Specifically, accuracy may be propagated differently depending on the accuracy factor used.

Remember that when evaluating semantic correctness metrics, if the key of a tuple is not accurate (does not correspond to a real-world entity) the whole tuple (all cells) are inaccurate. In other words, semantic correctness metrics evaluate if an attribute corresponds to the real world object represented by the key. For example, if a student identification number does not exists (e.g. stid 21), the data associated to such student id is incorrect too (i.e. ‘María Roca’ cannot be her name, ‘Carrasco’ cannot be her address, etc.). However, when evaluating syntactic correctness or precision, accuracy of cells does not depend on keys, for example, syntax constraints, belonging to a range or decimal precision can be checked independently of the key values.

Then, when joining two relations (two areas), the accuracy of the cells in the result is calculated differently for semantic accuracy. The cells in the result that are semantically correct are those that were obtained from a correct value with two correct keys. When one of the keys is incorrect, the complete tuple is incorrect too. So, the accuracy of a sub-area in the join result is calculated as the accuracy of the input sub-area multiplied by the accuracy of the keys of the other areas. However, when evaluating syntactic correctness or precision, the accuracy of cells in the result is the accuracy of the cell in the input relation.

In summary, we proceed as follows:

- For syntactic correctness and precision factors: Accuracy of a sub-area is estimated as the accuracy of the input sub-area.
- For semantic correctness: Accuracy of a sub-area is estimated as the accuracy of the input sub-area multiplied by the key accuracy of the other input areas.

For sub-areas containing join attributes (that were separated in previous step), accuracy is calculated separately from each input sub-area and the average is taken. If some sub-areas have the same accuracy value, they are fusioned in a unique sub-area. Finally, key accuracy is calculated multiplying accuracy of keys of all input areas.

Example 4.17. Continuing Example 4.16, we estimate semantic correctness of sub-areas, obtaining:

- Area QR_1 {stid < 101 \wedge year = 2005}; ? tuples; key accuracy = 0.33 (0.50 * 0.67); inputs: S_1 and M_2
 - Sub-area QR_{11} {name}; accuracy = 0.33 (0.50 * 0.67); input: S_{11}
 - Sub-area QR_{12} {year, mark}; accuracy = 0.33 (0.67 * 0.50); input: M_{21}
 - Sub-area QR_{13} {stid}; accuracy = 0.33 (average {0.50 * 0.67, 0.67 * 0.50}); input: S_{11} and M_{21}
- Area QR_2 {stid \geq 101 \wedge year = 2005}; ? tuples; key accuracy = 1.00 (1.00 * 1.00); inputs: S_2 and M_3
 - Sub-area QR_{21} {name}; accuracy = 0.87 (0.87 * 1.00); input: S_{22}
 - Sub-area QR_{22} {year, mark}; accuracy = 1.00 (1.00 * 1.00); input: M_{31}
 - Sub-area QR_{23} {stid}; accuracy = 1.00 (average {1.00 * 1.00, 1.00 * 1.00}); inputs: S_{21} and M_{31}

Sub-areas $\{QR_{11}, QR_{12}, QR_{13}\}$ and $\{QR_{22}, QR_{23}\}$ have the same accuracy value, so they are fusioned in sub-areas QR_{11} and QR_{22} respectively, obtaining:

- Area QR_1 {stid < 101 \wedge year = 2005}; ? tuples; key accuracy = 0.33; inputs: S_1 and M_2
 - Sub-area QR_{11} {stid, year, name, mark}; accuracy = 0.33
- Area QR_2 {stid \geq 101 \wedge year = 2005}; ? tuples; key accuracy = 1.00 (1.00 * 1.00); inputs: S_2 and M_3
 - Sub-area QR_{21} {name}; accuracy = 0.87
 - Sub-area QR_{22} {stid, year, mark}; accuracy = 1.00 \square

The accuracy of each rewriting is aggregated as the average of accuracy of cells, i.e. weighting the average of accuracy of sub-areas by the number of attributes projected in the sub-area.

Example 4.18. Continuing Example 4.17, the accuracy of QR_1 is 0.33 (the accuracy of its unique sub-area) and the accuracy of QR_2 is 0.97 (0.87 * 1 + 1.00 * 3). \square

Next sub-section explains the estimation of the number of tuples of each area.

4.3.3. Selectivity estimation

As the answer to a query can be built as the union of several rewritings, an accuracy value for the query is aggregated as the weighted sum of the accuracy of rewritings (weights are the number of tuples). To this end, we need to estimate how many tuples has each rewriting.

We propose to estimate the selectivity of rewritings in order to estimate their numbers of tuples. Any of the techniques discussed in Sub-section 3.3 can be used for estimating selectivity. In most cases, joins correspond to the equality between a primary key and a foreign key, so simple histograms are adequate for the estimation. However, other commercial algorithms used for query optimization or even statistics of previous execution of the same or similar queries can be used as estimation. In particular application scenarios, selectivity can be estimated by experts. Our approach is independent of the estimation strategy used but of course the obtained accuracy value depends on it.

We define selectivity as follows:

Definition 4.4 (selectivity). Given a query rewriting QR over a set of areas $\{R_1, \dots, R_k\}$, the *selectivity* of QR, denoted $sel(QR)$, is the proportion of tuples of the Cartesian product of the areas R_1, \dots, R_k that verify the rewriting predicates. If the query rewriting has no selection (nor join) predicates, then all tuples are returned and thus selectivity is 1. \square

The number of tuples of the rewriting is estimated multiplying the number of tuples of input areas by the selectivity of the rewriting.

Example 4.19. Continuing Example 4.18, consider that selectivity of QR_1 and QR_2 is estimated as $1/9$ and $1/8$ respectively. The complete metadata of the partition is:

- Area QR_1 $\{\text{stid} < 101 \wedge \text{year} = 2005\}$; 4 (6 * 6 * $1/9$) tuples; key accuracy = 0.33; inputs: S_1 and M_2
 - Sub-area QR_{11} $\{\text{stid}, \text{year}, \text{name}, \text{mark}\}$; accuracy = 0.33
- Area QR_2 $\{\text{stid} \geq 101 \wedge \text{year} = 2005\}$; 4 (4 * 8 * $1/8$) tuples; key accuracy = 1.00 (1.00 * 1.00); inputs: S_2 and M_3
 - Sub-area QR_{21} $\{\text{name}\}$; accuracy = 0.87
 - Sub-area QR_{22} $\{\text{stid}, \text{year}, \text{mark}\}$; accuracy = 1.00

The global accuracy value for $UserQuery_2$ is calculated weighting the accuracy of each rewriting by its number of tuples, obtaining $(4 * 0.33 + 4 * 0.97) / (4 + 4) = 0.65$. Note that the same value can be obtained weighting the accuracy of sub-areas of all areas by their number of cells, i.e. $(0.33 * 16 + 0.87 * 4 + 1.00 * 12) / (16 + 4 + 12) = 0.65$. \square

Next sub-section discusses the implementation of this evaluation strategy in an accuracy evaluation algorithm.

4.4. Reuse of the quality evaluation framework

In this sub-section we describe how the quality evaluation framework introduced in Chapter 3 is reused for data accuracy evaluation. We firstly describe the construction of quality graphs and then we present an accuracy evaluation algorithm.

4.4.1. Construction and adornment of quality graphs

Once the query rewritings have been generated, a quality graph can be build for representing the calculation of the user query as the union of several (possibly one) rewritings.

The graph is build as follows:

- The user query is represented by a target node
- Source relations are represented by source nodes.
- Areas of source relations are represented by activity nodes, called *area nodes*. Edges link each area node with the corresponding source node.
- Rewritings are represented by activity nodes, called *rewriting nodes*. Edges link each rewriting node with the ones representing the areas referenced by the rewriting.
- The union of rewritings (eventually only one rewriting) is represented by an activity node, called the *union node*. This activity is predecessor of the target node and successor of all rewriting nodes.

Figure 4.7 shows the quality graph for $UserQuery_2$ expressed as the union of rewritings QR_1 and QR_2 , computed in Example 4.13.

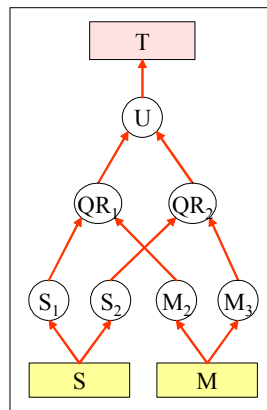


Figure 4.7 – Quality graph representing the union of two query rewritings

Several properties, necessities in the evaluation of data accuracy, adorn the quality graph and are input to the accuracy evaluation algorithm. Metadata describing areas and sub-areas is contained in the following property:

- *Areas*: It is an ordered list of records (representing areas) composed of the following fields:
 - *Predicate*: It is a conjunctive predicate defining the area
 - *Cardinality*: It is the estimated number of tuples in an area (verifying the area predicate)
 - *Key accuracy*: Accuracy of key attributes
 - *Sub-areas*: It is an ordered list of sub-areas, represented as pairs $\langle \text{attributes}, \text{accuracy} \rangle$ composed of: the set of attributes that defines the sub-area and its accuracy value.

This property is associated to source nodes (containing one record for each area) and area nodes (containing a unique record for the corresponding area). Rewriting nodes have associated further properties:

- *Predicate*: It is a conjunctive predicate defining the rewriting predicate
- *Projected attributes*: It is the set of attributes projected by the rewriting
- *Join attributes*: It is the set of natural join attributes of the rewriting
- *Consider key*: It is a Boolean flag indicating if key accuracy must be considered in accuracy propagation (This flag will be true for the propagation of semantic correctness and false for the propagation of syntactic correctness and precision).
- *Selectivity*: It is the selectivity of the rewriting

The evaluation algorithm calculates the following property:

- *Actual accuracy*: It is an estimation of the actual accuracy of data outgoing a node (*Aaccuracy* for short).

The algorithm also computes areas and sub-areas of the rewritings (and therefore associates the *areas* property to rewriting and union nodes). Next sub-section describes the evaluation algorithm.

4.4.2. Accuracy evaluation algorithm

In this sub-section we propose a basic algorithm for evaluating data accuracy. It follows the principle described in Sub-section 4.3, proceeding in three steps: (i) for each rewriting node, it creates an area, calculates its sub-areas and calculate property values (predicates, cardinality, key accuracy and accuracy of sub-areas); (ii) for the union node, it inserts areas of all rewriting nodes; and (iii) it aggregates accuracy values for all data edges. The pseudocode is sketched in Algorithm 4.2.

The first step performs two loops through area nodes incoming a rewriting. In the first loop, the *key accuracy* property is calculated multiplying the accuracy of input areas, the *cardinality* property is calculated multiplying rewriting selectivity by the cardinality of input areas and the *predicate* property is calculated as the conjunction of predicates of input areas and the predicate of the rewriting. The *insert* function of the *Predicate* class adds a new predicate as a new conjunction and also eliminates useless expressions (e.g. those that are less restrictive than other ones). In the second loop, the *sub-areas* property is calculated, inserting sub-areas of all input areas. This must be done in a separate loop because key accuracy (calculated in the first loop) is used for calculating accuracy of sub-areas in the case the *ConsiderKey* flag is switched on. After adding sub-areas to the unique area of the rewriting node, the *intersectAttributes* function intersects attributes of sub-areas with those projected by the rewriting, eventually eliminating empty sub-areas. Then, natural join attributes are separated in new sub-areas, calculating their accuracy as an average of accuracy of all sub-areas where they are contained. This is done by the *separateJoinAttributes* function. Finally, the *fusionSubAreas* function fusions sub-areas having the same accuracy value.

The areas created for each rewriting are added to a list (*Uareas*), so the second step only needs to set the list as value of the *areas* property of the union node.

In the last step, for each source or activity node, the *aggregateAccuracy* function performs a weighted sum of the accuracy of sub-areas, where weights are obtained multiplying the number of attributes of the sub-area by the cardinality of the area. The obtained value is associated to all data edges outgoing the node.

In next sub-section we deal with data accuracy improvement. We present a direct application of the approach for discarding the areas (or sub-areas) that cause overdrawing accuracy expectations.

```

FUNCTION ActualAccuracyPropagation (G: QualityGraph) RETURNS QualityGraph
  ListOfAreas Uareas; // will contain all areas, which will be associated to the union node

  FOR EACH rewriting node R in G DO // Set areas to writing nodes
    Area area;
    // First loop; sets predicate, cardinality and key accuracy
    INTEGER K= 1; // will contain the product of accuracies of all keys
    INTEGER card = G.getPropertyValue(R,"Selectivity");
    Predicate pred = G.getPropertyValue(R,"Predicate");
    FOR EACH predecessor A of R in G DO
      Area auxarea = G.getPropertyValue(A,"Areas").getFirstElement();
      K= K * area.getKeyAccuracy();
      card = card * area.getCardinality();
      pred.insert (area.getPredicate());
    ENDFOR;
    area.setKeyAccuracy(K);
    area.setCardinality(card);
    area.setPredicate(pred);
    // Second loop; sets sub-areas
    FOR EACH predecessor A of R in G DO
      Area auxarea = G.getPropertyValue(A,"Areas").getFirstElement();
      IF (auxarea.getConsiderKey() == TRUE)
        auxarea.updateSubAreasAccuracy(K);
        area.addSubAreas(auxarea.getSubAreas());
      ENDFOR;
    area.IntersectAttributes(G.getPropertyValue(R,"ProjectedAttributes"));
    area.SeparateJoinAttributes(G.getPropertyValue(R,"JoinAttributes"));
    area.FusionSubAreas();
    G.addPropertyValue(R,"Areas",{area});
    Uareas.add (area);
  ENDFOR;

  Node U = union node of G; // Set areas to the union node
  G.addPropertyValue(U,"Areas",Uareas);

  FOR EACH source and activity node A of G DO // Set accuracy aggregations to outgoing edges
    ListOfAreas areas= G.getPropertyValue(A,"Areas");
    INTEGER value= aggregateAccuracy (areas);
    FOR EACH data edge e outgoing A in G
      G.addProperty(e,"ActualAccuracy",value);
    ENDFOR;
  ENDFOR;
  RETURN G;
END

```

Algorithm 4.2 - Basic algorithm for propagating accuracy actual values

5. Accuracy improvement

In this section we discuss some base improvement actions for enforcing data accuracy when user accuracy expectations cannot be satisfied. Accuracy expectations correspond to upper bounds for the accuracy of result data. Remember that our evaluation approach groups result data in sub-areas having homogeneous accuracy. This means that we cannot assure that all cells in the result satisfy user expectations. Conversely, we can deliver data that, in average, satisfies user expectations.

The proposal for accuracy improvement is simple: *filtering “portions” of the query result having low accuracy*. In this sub-section we discuss several ways and several moments for performing such filtering, which depend on user expectations. Specifically, three types of accuracy expectations can be expressed:

- Accuracy of groups of cells, in average, should be lower than a threshold
- Accuracy of groups of tuples, in average, should be lower than a threshold
- Accuracy of the whole result should be lower than a threshold

The first type of condition is the most restrictive. It corresponds to users that cannot tolerate tuples having inaccurate values, even if other cells have high accuracy values. Furthermore, accuracy expectations may concern only certain attributes, for example, users may require telephone numbers to be accurate, ignoring the accuracy of other attributes. As we partition query result in sub-areas having homogeneous accuracy, this type of condition suggests the filtering of sub-areas having low accuracy. If the condition involves only certain attributes, only the sub-areas containing those attributes are candidate to be filtered. Note that such filtering can be done in an early phase of the evaluation, i.e. when adding areas to buckets. We use the term *selective rewriting* for naming this improvement action.

The second type of condition tolerates some attributes with low accuracy if the accuracy aggregation for the tuple is high enough. Certain users may accept receiving tuples that contain errors in some attributes if other cells are accurate. For example, when several attributes contain alternative ways of contacting customers (address, telephone, email...) errors in some attributes may be tolerated if the others have high accuracy values. This type of condition suggests the filtering of areas instead of sub-areas, i.e. aggregating an accuracy value from the accuracy of sub-areas and comparing it with user expectations. Conversely to the filtering of sub-areas, the filtering of areas must be done after computing rewritings, because the aggregation may concern sub-areas of several source relations.

The last type of condition does not care about the accuracy of attributes or tuples but expresses that the whole result should achieve a certain level of accuracy. Note that query results consisting of tuples with very low accuracy and other tuples with high accuracy, may be accepted. This corresponds to users that want as much result data as possible, without neglecting accuracy. For example, a user may send advertising letters to their customers, tolerating up to 5% of mail return due to errors in contact information; this means that he wants to send the maximum of letters while he can assure a certain benefice. In order to satisfy this kind of constraints, we should deliver as many data as possible without descending to much their accuracy. This suggests ordering areas according to their accuracy and aggregating accuracy values incrementally, stopping when accuracy expectations are no longer satisfied. Furthermore, we can incrementally deliver data to users, allowing them to stop the delivery when they have enough data or data has too many errors. This strategy can be performed only after aggregating accuracy of all candidate rewritings.

Three improvement actions were motivated: (i) selective rewriting of user queries, (ii) filtering of rewritings, and (iii) incremental delivery of results. In the following, we describe each action.

Selective rewriting consists in generating query rewritings that satisfy a certain quality requirement, specifically: ‘accuracy of all sub-areas must be higher than a threshold’. We propose modifying the rewriting algorithm, for excluding from buckets, the areas that have a “concerned” sub-area with lower accuracy. By *concerned* we mean that the sub-area provides attributes for solving the query, i.e. the query will project some of its attributes. As the accuracy of sub-areas was pre-calculated during partitioning, the implementation of this strategy is straight-forward.

The *filtering of rewritings* is also straight-forward. It can be done at the third step of our approach, i.e. when aggregating accuracy of rewritings (see Sub-section 4.3.2).

For the *incremental delivery of results*, we propose aggregating accuracy of rewritings and ordering them according their accuracy. A simple loop and two variables implement this improvement action, as sketched in Algorithm 4.3. The algorithm receives a list of areas (of rewritings) and an accuracy expected value and returns a

list of areas ordered by accuracy (descendent order). The returned areas are those that can be conveyed to users satisfying accuracy expectations; conveying more areas means not longer achieving user expectations.

Note that the estimation of rewriting selectivity is only useful for the third improvement action; the other actions filter individual sub-areas and rewritings.

```

FUNCTION IncrementalDelivery (inAreas: ListOfAreas, Eaccuracy: INTEGER) RETURNS ListOfAreas
    areas.sortByAccuracy();
    ListOfAreas outAreas;

    Area A = areas.getFirst();
    INTEGER acc= A.getAccuracy() * A.getCardinality(); // partial sum of accuracies * cardinalities
    INTEGER card = A.getCardinality(); // partial sum of cardinalities

    WHILE acc/card > Eaccuracy;
        outAreas.add(A);
        A = areas.getNext();
        acc = acc + A.getAccuracy() * A.getCardinality();
        card = card + A.getCardinality();
    ENDWHILE;
    RETURN areas;
END

```

Algorithm 4.3 – Incremental delivery of data

The following example summarizes the proposed improvement actions.

Example 4.20. Consider the four rewritings illustrated in Figure 4.8 (QR₁, QR₂, QR₃ and QR₄). Accuracy values are written under sub-areas and aggregated values for areas are written, in bold, under area name. Consider the following conditions:

- C₁) Accuracy of sub-areas should be greater than 0.75
- C₂) Accuracy of areas should be greater than 0.75
- C₃) Accuracy of query result should be greater than 0.75

Condition C₁ is only satisfied by QR₂. Furthermore, QR₂ is the unique rewriting that should be generated. For example, as A₁₂ does not satisfy the condition, the area containing A₁₂ is deleted from the corresponding bucket and therefore, the rewriting QR₁ is not generated.

Condition C₂ is satisfied by QR₁ and QR₂. Although sub-area A₁₂ has a lower accuracy value, the accuracy aggregation for the rewriting is acceptable. For that reason, filtering must occur after accuracy aggregation.

In order to verify condition C₃, rewritings are ordered according to their accuracy, obtaining the following order: QR₂ > QR₁ > QR₃ > QR₄. We incrementally aggregate accuracy of such rewritings. For {QR₂} accuracy is 0.90. For {QR₂, QR₁} accuracy is $(0.90 * 20 + 0.80 * 10) / (20 + 10) = 0.86$. Analogously, for {QR₂, QR₁, QR₃} accuracy is 0.80. For {QR₂, QR₁, QR₃, QR₄} accuracy is 0.70, which does not satisfy the condition; so QR₄ is eliminated. □

The most restrictive are the conditions, the smallest is the result. Accuracy expectations should be balanced with completeness expectations for avoid filtering too much data and conveying a representative set of tuples to users. The trade-off among data accuracy and other quality factors is discussed, in Chapter 6, as future work.

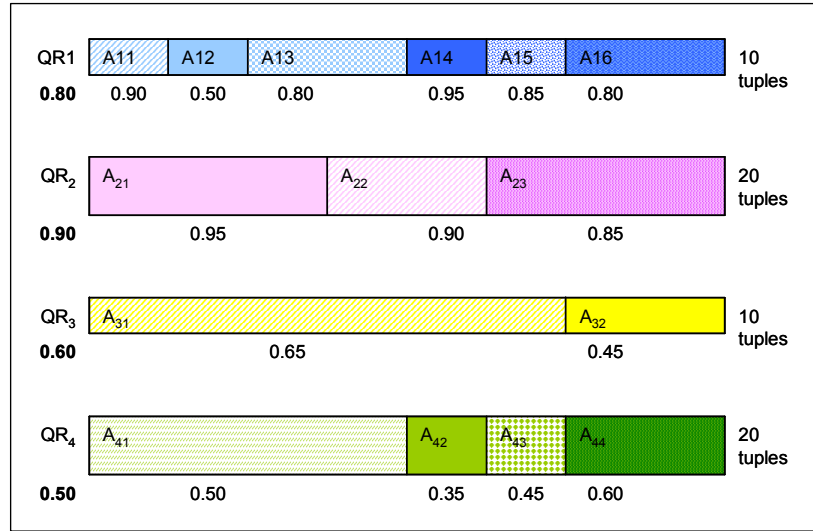


Figure 4.8 – Filtering areas and sub-areas

In terms of the elementary improvement actions proposed in Sub-section 4.4 of Chapter 3, the improvement actions proposed in this section correspond to deleting portions of the quality graph, i.e. invoking the *replaceSubGraph* action. In addition, the proposed improvement actions can be combined with other actions not treated in this thesis, for example, error correction techniques, in order to implement more complex improvement strategies for specific DISs.

6. Conclusion

In this chapter we dealt with data accuracy evaluation and enforcement topics.

Based on the partitioning algorithm proposed in [Rakov 1998], we proposed the partitioning of source relations according to data accuracy and the propagation of such partitions to query results in order to better describe the distribution of inaccuracies. User queries are rewritten in terms of partitions and accuracy values are aggregated for the different rewritings. The query result consists in the union of tuples returned by rewritings. The number of tuples is calculated, estimating the selectivity of the rewriting, in order to aggregate an accuracy value for the query result.

We presented a basic algorithm for data accuracy evaluation, which implements the proposal. Compared to existing proposals for *a priori* evaluation, our algorithm explicitly indicates the areas that have lower accuracy, which allows filtering data not satisfying accuracy expectations. The precision of the estimation relies on the techniques used for measuring accuracy of source relations, partitioning them and estimating the selectivity of rewritings. We do not tested different techniques in order to suggest the utilization of specific ones. This topic is suggested as perspective in Chapter 6.

We also proposed some basic improvement actions, for filtering data with low accuracy in order to satisfy several types of accuracy expectations. Other improvement strategies can be analyzed and combined to these ones in order support more complete improvement strategies. The partitioning mechanism and the evaluation algorithm proposed in this section may help in the analysis. The proposal can be used at different phases of the DIS lifecycle (e.g. at design, production or maintenance phases), either for communicating data accuracy to users, comparing data sources, checking the satisfaction of user accuracy expectations or analyzing improvement actions for enforcing data accuracy. Chapter 5 illustrates the use of the approach in a real application.