1

A synopsis based approach for XML fast approximate querying

Sara Comai, Stefania Marrara, and Letizia Tanca

Politecnico di Milano, Dipartimento di Elettronica e Informazione
Piazza L. Da Vinci 32, I-20133 Milano, Italy
{comai, marrara, tanca}@elet.polimi.it

Summary. *XML was born to represent, exchange and publish information on the Web, but now it has spread in many other applications. Due to this success, the W3C has proposed a new query language, XQuery, specifically designed to query XML data. XQuery allows to obtain exact answers to queries; however when applied to large XML repositories or warehouses, such precise queries may require high response times. Our research proposes a methodology for the semi-automatic derivation of summarized documents (synopses) for massive, heterogeneous XML data-sets, with the final aim of producing query transformation rules from queries on the original data-sets to queries on the summarized data-set.*

1.1 Introduction and Motivation

In the last few years, XML has spread in many application fields and today it is used as a format to exchange data on the web, to ensure interoperability among applications. Due to this success, the W3C has proposed a new query language, XQuery [W3C04], specifically designed to query XML data. XQuery is a well-defined but rather complex language [HPG04]. In this work we propose a new approach to overcome the problem of the high computational costs required by aggregate queries over massive XML data collections. In traditional relational warehouses [GPA+98] a similar problem is solved by means of fast *approximate* queries, that use concise data statistics based on histograms or on other statistical techniques. Their most common application is for aggregate queries in modern decision support systems, where large volumes of data need to be queried, and quick and interactive responses from the DBMS are claimed, e.g., to analyze the data in the warehouse in order to get trend information to evaluate marketing strategies. In such applications, users are often more interested to obtain an approximate answer computed in a short time rather than an exact one obtained in some minutes or, at the worst, hours.

In this work we show how to extend one of such approaches also to query massive XML data sets, to obtain approximate answers in very short computation times.

The basic idea for approximate answers is to store pre-computed summaries of the XML warehouse, also called *synopses* (concise data collections), and to query them instead of the original database, thus saving time and computational costs. For this reason we attempt at obtaining a synopsis with a structure as similar as possible to the original one in order to combine conciseness of the data set and easiness of the query formulation. In our scenario, the user should pose a query to the system, which should use the synopsis instead of the original data to answer the query. Consequently, the answer should come within a much shorter time to the price of a loss in precision. The research work's purpose involves techniques for such transparent query transformation, as well as evaluation methods for estimating the error produced by the evaluation of the query on the synopsis instead of the original data.

The structure of the work is as follows. Section 1.2 introduces a general overview of the full approach and an example of XML collection that will be used during the exposition of the approach. Section 1.3 presents some basic definitions. In Section 1.4 we describe the structure of the synopsis and define how to automatically create the XQuery query to construct the synopsis of a given structure. Section 1.5 describes how to transform a query in XQuery on the original data set into the corresponding query on the synopsis. Section 1.6 contains considerations about the quality of the approach. Section 1.7 shows the results of some experiments generated by means of a prototype tool, Section 1.8 presents some literature works about synopses, and, finally, in Section 1.9 we outline our conclusions.

1.2 Overview of the approach

Fig.1.1 shows the basic steps of our approach: initially, a collection of XML documents, sharing a unique *DTD* or *schema*, is collected in a smaller group of XML documents, each document representing a summarization of the data contained in the original collection by means of statistical techniques. This new collection is called *synopsis* and the summarization is obtained by means of a XQuery transformation TR from the original data. The XML synopsis conforms to a new schema (e.g., DTD) generated from the original schema by a step by step XQuery transformation.

When the user poses an aggregate query Q, over the original XML collection, his/her query is transformed by a XQuery transformation QTR into a query to the synopsis collection Q'. Finally, the answer (probably approximate) is computed and returned to the user.

In the general case, the collection of XML documents we suppose to deal with is composed by several different groups, each of them sharing the same

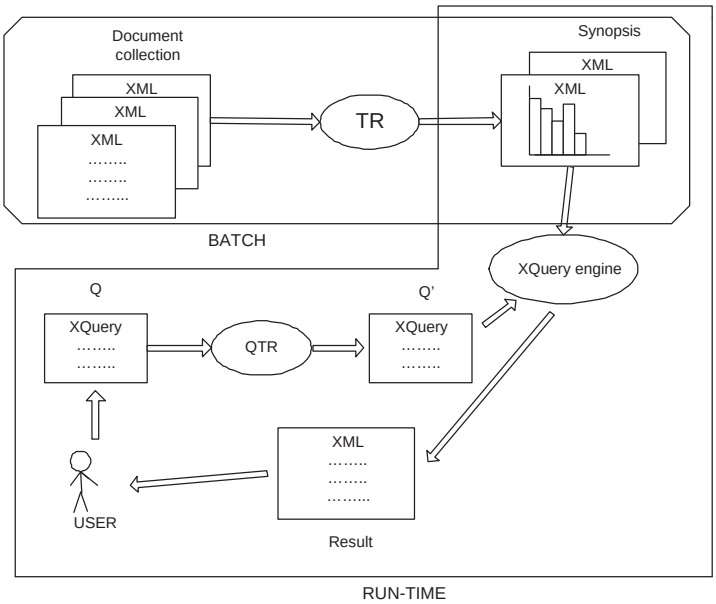

Fig. 1.1. The XQuery approach for XML synopsis

DTD or the same XML Schema. In the sequel we will use the word *schema* referring both to DTD and XML Schema. In our approach, each group of documents is summarized using a statistical technique creating one synopsis. The union of all the synopses generated from each group of documents composes the entire synopsis of the XML document collection. We use only one summarizing technique for the entire data collection (e.g., we use the equi-width histograms), so that it is possible to use just one implementation of the statistical aggregate functions to query the resulting synopsis. This choice motivates why in this work we prefer to deal with DTDs and not XML Schemas: DTDs are a weaker formalism than XML schema but they give the necessary knowledge about the structure of the XML document tree. Since we do not use different statistical techniques inside the same synopsis, we do not really need a deep knowledge of the data types used in the original data collection and then we do not need the more complex XML Schema formalism. Moreover, there exist many collections of data, that could have benefit from our methodology, that do not have a real schema and the DTD is created a posteriori. Naturally, the choice of the statistical technique is application dependent and can be different for each collection we analyze, then, nobody denies that in some applications XML Schemas could be more useful than DTDs; in this work, we focus on histograms because they seem the most suited statistics for the mixed content of a generic XML document, which typically stores categorical elements (e.g., names or colors etc.) and numerical elements (e.g.,

ages or prices) together in the same document. Indeed, it is possible to use wavelets to summarize only numerical data, while samples work well with uniform distributions that are not common in XML.

1.2.1 Running example

During the exposition of the approach, we consider the following *schema* (DTD) describing a car store: cars are characterized by color and selling details including model, customer's city and price and optional features of the sold car. Fig.1.2 shows a sample XML document conforming to such *schema*.

```
<? xml version = "1.0" ?>
<!ELEMENT list (car+)>
<!ELEMENT car(selling, color)>
<!ELEMENT selling(details)>
<!ELEMENT details(model, city, price, optional*)>
<!ELEMENT model (PCDATA)>
<!ELEMENT color (PCDATA)>
<!ELEMENT city (PCDATA)>
<!ELEMENT price (PCDATA)>
<!ELEMENT optional (PCDATA)>
```

1.3 Representing summarized XML data

Following a common use, we represent a XML document as a tree $T = (V, E)$, where V is the node set comprising both nodes representing tags and nodes representing text content and attributes. Attributes are not explicitly handled in this work because, if a literal semantics [GMW99] for the representation of XML documents is adopted, they can be treated as a particular case of PCDATA elements. $E \subseteq V \times V$ represents elements and text containment arcs. The target document collection can be composed by groups of documents having different schemata, and, in this case, the proposed approach is applied separately to each group, while the final synopsis is the union of all the synopses generated by the different groups.

We represent XML elements with their paths from the root of the document they belong to using a XPath 1.0 [W3C99] notation, in order to distinguish elements with the same tag name but with different internal meaning (e.g., a person's home address is different from the address of the company the person belongs to).

The entire methodology to create an XML synopsis can be described by the following steps:

- Initially, the designer investigates the target data collection, in order to identify the most common aggregate queries in the application. On the

```
<list>
  <car>
    <color> white </color>
    <selling>
      <details>
        <model> Fiat Brava </model>
        <city> Milano </city>
        <optional> ABS </optional>
        <optional> airbag </optional>
        <optional> electronic closure key </optional>
      </details>
    </selling>
  </car>
  <car>
    <color> white </color>
    <selling>
      <details>
        <model> Fiat Brava </model>
        <city> Roma </city>
        <optional> ABS </optional>
      </details>
    </selling>
  </car>
<car>
    <color> white </color>
    <selling>
      <details>
        <model> Opel Corsa </model>
        <city> Milano </city>
        <optional> airbag </optional>
        <optional> electronic closure key
</optional>
      </details>
    </selling>
  </car>
  <car>
    <color> white </color>
    <selling>
      <details>
        <model> Fiat Marea </model>
        <city> Milano </city>
      </details>
    </selling>
  </car>
</list>
```

Fig. 1.2. A sample XML document

basis of this set of queries, he/she identifies a collection of frequent queried aggregations and the parameters of each histogram involved;

- Then the synopsis is automatically created by a XQuery engine on the basis of the structure decided in the previous step;
- The data collection is ready for querying: each query is transformed and redirected to the synopsis, obtaining an answer (possibly approximate) in a much shorter time than querying the original data set.

At the beginning of the design of a synopsis, we identify *the descriptor of the synopsis* as a collection of frequent queried aggregations. The most natural set of synopses for an approximate query engine would include an aggregate element for each leaf element in the document tree. We refer to documents containing those aggregate elements as *base synopses*. We would like to evaluate the relationship between two leaf elements in the tree by means of some kind of combination of the base synopses of the target elements. This approach is similar to the problem of evaluating a join between two relations trying to combine the base synopses of the relations themselves. This problem has been discussed in literature in [AGPR99]: in this work they prove that the use of base samples to estimate the output of a join of two or more relations can produce a poor quality approximation. The reasons that motivate this claim are also valid in the histogram domain:

- **non-uniform result distribution:** in general, the join of two histograms does not represent the real data distribution of the output of the join, since each original data item can be collected into one or more histograms depending on the dimensions chosen to build the statistics;
- **inaccurate result compositions:** the join of two histograms typically composes bucket frequencies instead of the actual data. This can lead to both inaccurate answers and very poor confidence bounds since they compose the errors committed when we consider a frequency value instead of the real data value.

These considerations motivate our choice to construct the set of synopses taking into account the most common aggregate queries and the most interesting relationships among the elements of the document structure. In this scenario each synopsis can answer a limited (but interesting to the user) set of queries but the accuracy is guaranteed.

Definition 1 *[Synopsis descriptor] The descriptor of a XML synopsis is defined by a set of pairs* H_s=$\{(summ\_e, < crit\_g_1, \ldots, crit\_g_n >)\}$, *where*

- *summ_e is the path expression of the element to be summarized, and*
- $< crit\_g_1, \ldots, crit\_g_n >$ *(i.e., grouping criteria) is a (possibly empty) sequence of path expressions of the elements whereof the element given by summ_e is grouped in the summarization process.*

As histograms are constructed over elements and not over paths, in the sequel we will use the name *summ_e* just to indicate the leaf element of the path named *summ_e* and *crit_g* for the leaf element of the path *crit_g*.

As an instance, referring to the running example, suppose that the user be interested in storing a summarized collection about the models of the cars w.r.t. the color and the city where the cars were sold. In this case, H_s is represented as follows:
H_s=$\{$(`summ_e`, <`crit_g`>)$\}$=$\{$(`list/car/selling/details/model`, <`list/car/color`,`list/car/selling/details/city`>)$\}$.

Once the descriptor of the synopsis document has been defined, another important decision is the set of parameters (e.g., number of buckets or boundary values) of the histogram that will store the data of *summ_e*.

Definition 2 *[parameter document] The parameter document P is an XML document containing the boundaries of the buckets of the histogram to be obtained by the summarization described by a pair* $(summ\_e, < crit\_g_1, ..., crit\_g_n >)$. *The document can have the following structure:*

- *if summ_e ends with a categorical element (e.g., the element* <`color`>*) an interval content of a sample bucket of its histogram has the form* $<$ *interval* $><$ *bv* $>$ *value* $<$ */bv* $><$ */interval* $>$, *where bv stands for boundary value;*

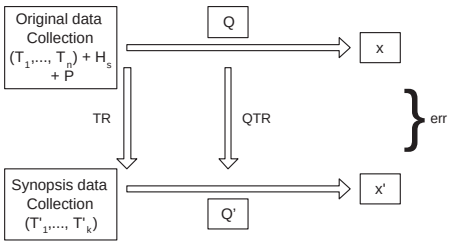

Fig. 1.3. Transformation diagram between the original data collection and the corresponding synopsis, and between the aggregate queries, exact and approximate. In the figure, x and x' are the result of the original query and the result of the approximate one respectively.

- *if summ_e ends with a numerical element, the interval can have the form* $< interval >< bv_{min} > value < /bv_{min} >< bv_{max} > value < /bv_{max} >< /interval >$*)*

For example, if *summ_e* ends with the element <`color`> (a categorical element) of the running example, an interval content of a sample bucket of its histogram has the form $< interval >< bv > blue < /bv >< /interval >$, while if *summ_e* ends with a numerical value, the interval can have the form $< interval >< bv_{min} > 0 < /bv_{min} >< bv_{max} > 99 < /bv_{max} >< /interval >$.

The correspondences between the original data collection and the corresponding synopsis collection, between the original query and the query on the synopsis, and their results are shown by the diagram of Fig. 1.3. In the figure, the original query Q applied to the data collection produces an exact result x; moreover a transformation rule QTR computes the query Q' that, applied to the synopsis created by the transformation rule TR, produces a new result x', which may be different from x. This difference is expressed by the approximation error err, which will be described in the sequel.

Given H_s and P, the XQuery transformations involved are expressed as follows:

Definition 3 *[TR] Given a set* $\{T_1, \ldots, T_n\}$ *of XML documents (where* $\{T_1, \ldots, T_n\}$ *share the same* schema*), we call* TR *:* $(\{T_1, \ldots, T_n\}, H_s, P) \rightarrow \{T'_1, \ldots, T'_k\}$*,* $k \leq n$ *(often it will be* $k \ll h$*) the transformation able to construct the* XML synopsis*(*DATA_{syn}*).* $\{T'_1, ..., T'_k\}$ *is a (small) set of XML documents conforming to one new* schema*, which we call* synopsis schema*.*

Definition 4 *[SchemaTransf] Let* $\{S_1, ..., S_n\}$ *be the set of* schemata *of the target XML data-set. We call* SchemaTransf *:* $(\{S_1, ..., S_n\}, H_s) \rightarrow \{S'_1, \ldots, S'_k\}$ *the transformation on the* schemata *of the initial set of documents to construct the corresponding* synopsis schemata*,* $\{S'_1, \ldots, S'_k\}$*.*

Definition 5 *[QTR] Let* $\{Q\}$ *be a set of aggregate queries on the original data set and* $\{Q\prime\}$ *be the set of corresponding synopsis queries. Then,* QTR :

$\{Q\} \rightarrow \{Q\prime\}$ *is the transformation that returns, for each original query* $Q \in \{Q\}$*, a new query* $Q\prime \in \{Q\prime\}$*.*

If we consider the aggregate queries, let $x \in \mathcal{R}$ be the exact numerical answer of an aggregate query Q and let $x' \in \mathcal{R}$ be the corresponding estimated answer on the synopsis. Then, the approximation error of Q can be measured in absolute, relative or combined terms [MVW98] as follows:

- *Absolute error* of a query:
  $err^{abs} = |x - x'|$.
- *Relative error* of a query:
  $err^{rel} = \frac{err^{abs}}{x} = \frac{|x-x'|}{x}$, for $x > 0$.
- *Combined error* of a query:
  $err^{comb} = min\{\alpha * err^{abs}, \beta * err^{rel}\}$, where α and β are positive constants, used to tune the relative importance of the two errors one w.r.t. the other. If $x = 0$, then $err^{comb} = \alpha * err^{abs}$.

For a more detailed explanation about error approximation measures see [MVW98]. In Section 1.6 we apply these definitions to our methodology and study the degree of approximation obtained querying our synopses instead of the original data.

1.4 Structure of the synopsis

In this section we describe the methodology for the design and construction of a synopsis for a given set of XML documents. To simplify the synopsis construction we suppose that for each element $e \in H_s$ a single XML synopsis document be obtained.

Since we consider documents containing both categorical and numerical data, in this work we summarize the XML data collection by means of equi-width histograms. Equi-width histograms group contiguous ranges of the element values into B buckets with the criterion that the sum of the spreads of the values in one bucket is approximately equal to $1/B$-times the sum of the spreads of all values. In order to construct the equi-width histograms of the synopsis the designer must fix the boundary values of each bucket, deciding the content of the parameter document P. To describe P we use the following definition:

Definition 6 *[Active Domain] We call* active domain *D of an element e the set of distinct element values of the domain of e that actually appear in the target XML document collection.*

The synopsis histogram is constructed according to the following rules:

- Given $e \in H_s$, the histogram collects the data represented by *summ_e*;
- if *summ_e* is a leaf node, each boundary value represents

  - a value of the *active domain* of *summ_e* leaf element if this is a categorical element (e.g., color or city),
  - an interval in the *active domain* of *summ_e* leaf element if it holds a numerical value.
- If *summ_e* is a non-leaf element, and $A_1, \ldots, A_n$ are the active domains of the descendant leaf elements of *summ_e*, then the *summ_e* histogram holds the Cartesian product
  $\mathcal{BV} = A_1 \times \ldots \times A_n$;
- the frequency value $freq$ represents the number of *summ_e* values that satisfies the grouping conditions expressed by $< crit\_g_1, \ldots, crit\_g_n >$ and belongs to the bucket defined by the boundary value.
- if we consider histograms of numerical data, the buckets must not overlap in order to avoid that the same item of the document collection be counted twice in the histogram.

As an example, referring to the schema of the running example in Sect.1.2.1 consider the synopsis where *summ_e* is $list/car/color$, i.e., a categorical leaf element. In this case, let us suppose that the boundary values be *blue, red, black* and *grey*, hence the parameter document P is shown in Figure 1.4:

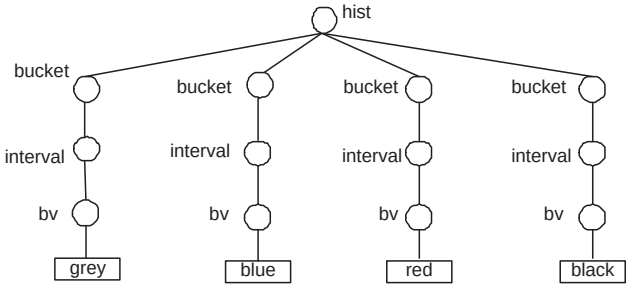

Fig. 1.4. The parameter document for a categorical element histogram

If *summ_e* is $list/car/selling/details/price$, i.e., a numerical leaf element, the boundary values can range from \$5000 of the cheapest car to \$65000 of the most expensive one sold in the store. Therefore, a possible parameter document P of the histogram of this element is shown in Figure 1.5:

Using the equi-width histograms, the parameter document of a numerical element can be automatically constructed providing as parameters the total range of the possible values of the elements (e.g., 5000 to 65000) and the number of buckets to construct (e.g., 6).

If *summ_e* is $list/car/selling/details$, i.e., a non-leaf element, the values to be considered in the histogram are obtained as combinations of the leaf descendant values $list/car/selling/details/model$, $list/car/selling/details/city$, $list/car/selling/details/price$, and $list/car/selling/details/optional$. For

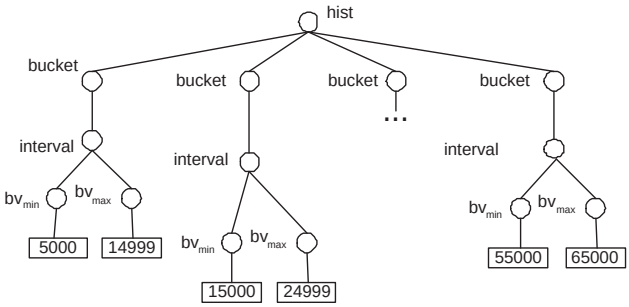

Fig. 1.5. The parameter document for a numerical element histogram

example, one possible value for detail is the tuple $(FiatBrava, Rome, \$12000, ABS)$. Let us now consider three possible situations:

- all the leaf descendant of *summ_e* have cardinality (1:1) w.r.t. *summ_e*. In this case a possible parameter document P is shown in Figure 1.6:

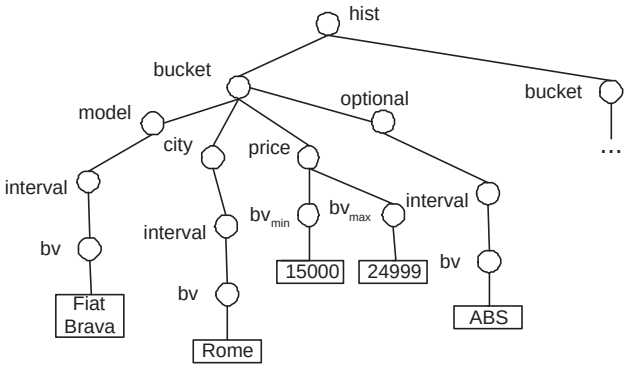

Fig. 1.6. The parameter document for an element having leaf descendants with cardinality 1:1

  Note that the path nodes between *summ_e* and the leaf descendant values considered as boundary value of the histogram appear in each bucket structure between the node $< bucket >$ and the node $< interval >$.
- if one or more leaf descendant of *summ_e* are optional elements (cardinality (0:1)), we overcome the problem by adding to the active domain of the optional element/s the value ND (i.e., Not Determined) in order to store the elements that do not contain the optional element in the document. As an example, let us suppose that *summ_e* is $list/car/selling/details$ and *city* is optional with active domain $A = \{Rome, Milan, Verona\}$.

The new active domain of *city* used for constructing the histogram is $A = \{Rome, Milan, Verona, ND\}$, therefore a possible parameter document will contain also the buckets dealing with the absence of a *city* value in some documents of the original collection (see, as instance, Figure 1.7).

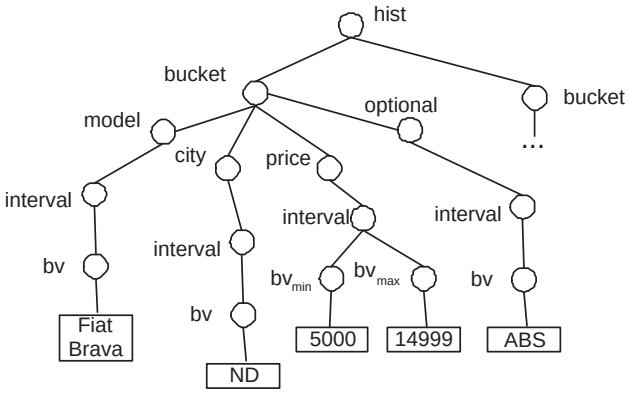

Fig. 1.7. The parameter document for an element having leaf descendants with cardinality 0:1

- if one or more leaf descendant of *summ_e* can appear more than once in *summ_e* sub-tree (cardinality (0:n)) then we note an explosion in the number of the buckets composing the histogram. Indeed, we should consider any possible combination of the children elements values of *summ_e* taking into account the possibility that the same element can appear more than once in the same document. Therefore this case is deprecated, and we strongly suggest not to choose as *summ_e* an element containing children elements with cardinality n. A possible solution is to choose as summarized element one of the children of the element we would like to summarize with cardinality 1:1 w.r.t. *summ_e*. As instance, consider the P document fragment in Fig. 1.8 constructed supposing that the same car model can be sold in Rome or in Milan or in both towns:

Obviously, if we consider a more complex case where the number of possible cities can be $n >> 2$, then the number of buckets in each histogram (one for each model) increases more than exponentially.

1.4.1 XQuery rules for synopsis computation

We now define the transformation TR to construct the standard synopsis from the original data collection. There are two main cases:

- construction of the histogram of a leaf element (leaf *summ_e*);
- construction of the histogram of a non leaf element (non leaf *summ_e*);

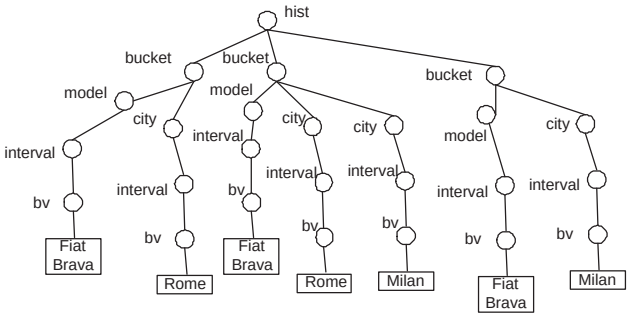

Fig. 1.8. The parameter document for an element having leaf descendants with cardinality 0:N

Case of leaf summarized element

The histogram of each $e = (summ\_e, < crit\_g_1, \ldots, crit\_g_n >)$, where $summ\_e$ is a leaf element, can be computed using the XQuery code Q_{hist}, where `www.doc.com` is a URI containing the set of documents to summarize, while `www.parameters.com` specifies the *parameter document* P of the element to summarize. The code $Q(hist)$ is the following:

1.<hist>
2. LET $V := document("www.doc.com")/summ_e
3. FOR $b IN document("www.parameters.com")/interval
4. RETURN
5. <bucket>
6. <interval>
7. FOR $$bv$$ in $$b//bv_{min}$$
8. $$< bv_{min} >$$ $$b//bv_{min}$$[text()] $$< /bv_{min} >$$
9. $$< bv_{max} >$$ $$b//bv_{max}$$[text()] $$< /bv_{max} >$$
10. FOR $$bv_2$$ in $$b//bv$$
11. <bv> $$bv_2$$[text()] </bv>
12. </interval>
13. <freq> count(IF $$/interval//bv_{min}$$
14. THEN $V[text()>$$b//bv_{min}/text()$$ AND
15. text()<$$b//bv_{max}/text()$$]
16. ELSE $V[text()=$$b//bv/text()$$])
17. </freq>
18. </bucket>
19. </hist>

The construction of the histogram is based on two FOR clauses (lines 7 and 10), selecting for each bucket the appropriate boundary values (numerical or categorical); then, function `count` in line 13 computes the frequency of the bucket represented by the chosen boundary value(s).

Case of non-leaf summarized element

If *summ_e* is a non-leaf element the Cartesian product of its descendant leaf elements must be computed; indeed the tuple of values contained into its descendant leaf elements represents the content value of *summ_e*: Q_{hist} becomes more complex. Indeed, in this case we do not have a fixed XQuery query as in the previous case but we need a function able to construct the query case by case. In the following we show the pseudo code of this function, named `create_hist`. Inputs of the function are the element to be summarized *summ_e*, the grouping criterion sequence <*crit_g*>, the list of leaf children of *summ_e*, the target document collection *doc* and the histogram parameter document *P*. The function returns the query string for the construction of the histogram of *summ_e*.

```
string create_hist(summ_e, <crit_g>, leaf_children(summ_e),
doc, P)
{
  string q, t;
  element el;
  q.insert("<hist> LET $V:=document(",doc,"/",summ_e);
  q.insert(endline);
  q.insert("FOR $b IN document(", P,")/interval", endline,
                    "RETURN <bucket><interval>",endline);
  while(leaf_children(summ_e) is not empty)
   { el = first element of leaf_children(summ_e);
     if (el is a numerical element)
     {q.insert("<el>", endline);
      q.insert("FOR $bv IN $b//",el,"/bv_min", endline,
       "<bv_min>$b//",el,"/bv_min[text()]</bv_min>",endline,
      "<bv_max"$b//",el,"/bv_max[text()]</bv_max>",endline);
         t.insert("text()>$b//",el,"/bv_min[text()]AND
                        text()<$b//",el,"/bv_max[text()]");
         remove el from leaf_children(summ_e);
         if(leaf_children(summ_e) is not empty)
           t.insert("AND");
     }else if(el is a categorical element)
      {q.insert("<" el ">", endline);
       q.insert("FOR $bv2 IN $b//",el,"/bv", endline,
                  "<bv>",bv2,"[text()]</bv>",endline);
            t.insert("text()=$b//",el,"/bv[text()]");
            remove el from leaf_children(summ_e);
          if(leaf_children(summ_e) is not empty)
           t.insert("AND");
       }
       q.insert("</",el,">",endline);
   }
  q.insert("</interval>",endline);
  q.insert("<freq> count($V[",t,"])</freq>",endline);
```

```
  q.insert("</bucket>",endline,"</hist>");
}
```

The function `create_hist` uses two strings, q and t: q stores the query, while t stores the boundary values of each bucket in order to compute the frequency of the bucket itself. For each bucket, the while cycle creates the structure of the bucket and stores the values of the boundary values given in P in the string t, which is used at the end of the function to compute the frequency. For example, the query constructed to create the histogram of the element *list/car/selling/details* (with children *model*, *city*, *price* and *optional*) is:

```
<hist> LET $V:=document("www.doc.com")/list/car/
selling/details
FOR $b IN document("www.parameters.com")/interval
 RETURN <bucket><interval>
  FOR $bv2 IN $b//model/bv
  <model>
   <bv> $bv2[text()]</bv>
  </model>
  FOR $bv2 IN $b//city/bv
  <city>
   <bv> $bv2[text()]</bv>
  </city>
FOR $bv IN $b//price/bv_min
  <price>
  <bv_min>$b//price/bv_min[text()]</bv_min>
  <bv_max>$b//price/bv_max[text()]</bv_max>
  </price>
  FOR $bv2 IN $b//optional/bv
  <optional>
   <bv> $bv2[text()]</bv>
  </optional>
</interval>
<freq> count($V[text()=$b//model/bv[text()]AND
text()=$b//city/bv[text()]AND
text()>$b//price/bv_min[text()]AND
          text()<$b//price/bv_max[text()] AND
          text()=$b//optional/bv[text()])
</freq>
</bucket> </hist>
```

Construction of the synopsis document structure

To define the transformation for computing the whole synopsis, we first need a preliminary definition:

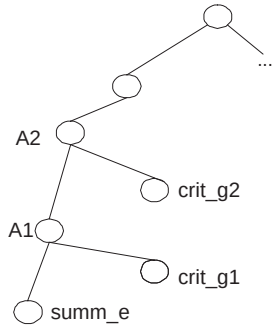

Fig. 1.9. LCA example structure

Definition 7 *[Lowest Common Ancestor]* $\forall i \in \{1..n\}$*, the lowest common ancestor (LCA)* A_i *is the deepest node that is ancestor both of summ_e and of the elements in* $< crit\_g_i, \ldots, crit\_g_n >$*,where*

- *if* $i \in [1, ..., n-1]$ *then* $< crit\_g_i, \ldots, crit\_g_n > \subseteq < crit\_g_1, \ldots, crit\_g_n >$ *else*
- $< crit\_g_i, \ldots, crit\_g_n > = < crit\_g_n >$ *if* $i = n$.

Fig. 1.9 shows an example of tree of a synopsis and the LCAs of *summ_e* and $< crit\_g_1, crit\_g_2 >$. In this figure we can see that $A1$ is the LCA of *summ_e* and $crit\_g_1$, while $A2$ is the LCA of *summ_e* and $< crit\_g_1, crit\_g_2 >$.

Note that, by construction, each common ancestor A_i is the root element of the smallest subgraph that contains the histogram derived from one value of the *active domain* of $crit\_g_i$. Since we have n histograms to be stored, it follows that we need exactly n subgraphs and consequently A will be repeated n times in the synopsis document; thus the following theorem holds:

Theorem 1 *Given the elements summ_e and crit_g, the corresponding LCA A is repeated inside the synopsis document n times, where* $n = |D|$ *and D is the active domain of crit_g.*

The synopsis histogram is constructed from the original data collection starting from the set of paths defining the histogram itself (*summ_e*, $< crit\_g_1, \ldots, crit\_g_n >$) and the tuple of parameters $\in P$. The construction of the synopsis is based on a set of rules that define the transformation TR and have been detailed in [Mar05]. In this work we show an example of XQuery code obtained to collect the models (representing *summ_e*) w.r.t. color and city (representing $< crit\_g_1, crit\_g_2 >$) from the running example of Section 1.2.1.

```
1. <list>
2. FOR $g1 IN LIST/CAR/COLOR
3.  <car> \* first common ancestor of
```

```
                    color and model*\
4.   <color> $g1 </color> \* first crit\_g*\
5.   <selling>
    \* element that connects A1 and A2*\
6.   FOR $g2 IN
     LIST/CAR/SELLING/DETAILS/CITY
 \*the LET is inside the most nested FOR*\
7.   LET $V:=www.doc.doc/LIST/CAR/MODEL
8. [../CAR/COLOR=$g1 and
   ../CAR/Y/X/CITY=$g2] \*binding of the
                        crit\_g criteria*\
9.   RETURN
10.   <details> \* A2 *\
11.    <model> - histogram - </model>
12.    <city> $g2 </city> \* last
crit\_g *\
13.   </details>
14.  </selling>
15. </car>
16. </list>
```

An example of the rules that define TR is the following:

TR-Rule 1 *The query body is composed by n nested FOR clauses* ($n = |\{crit\_g\}|$)*, following the same order of the elements as in* $< crit\_g_1, \ldots, crit\_g_n >$.

Each FOR clause generates a branch in the synopsis tree ready to store the histogram of the summarized element path *summ_e* w.r.t. the condition expressed by *crit_g*. The structure is recursive, because each branch born from the conditions
$crit\_g_i \in < crit\_g_1, \ldots, crit\_g_n >$, $i = 1, \ldots, n$, is divided into n new branches, one for each *active domain* value of $crit\_g_{i+1} \in < crit\_g_1, \ldots, crit\_g_n >$. In this way each sub-branch can store the histograms of the data which respect both $crit\_g_i$ and $crit\_g_{i+1}$ etc. In the example code this rule creates the FOR clauses in the lines 2 and 6 of the query. A sample synopsis document constructed for a document collection respecting the structure of the DTD in Section 1.2.1 is shown in Figure 1.10. The structure of the synopsis chosen for this example H_s is a collection of the models of the cars w.r.t. the color and the city where the cars were sold. Therefore, H_s is represented as follows:
H_s={(`summ_e`, <`crit_g`>)}={(`list/car/selling/details/model`,
<`list/car/color`,`list/car/selling/details/city`>)}.

The other rules describe what appear in the clauses of the query, how many FOR cycles the query needs depending on the structure of the document tree, which branches of the document tree have to appear in the synopsis structure, and in which point of the synopsis tree the histograms are stored (see the details in [Mar05]).

```
<list>
 <car>
   <color> white </color>
   <selling>
    <details>
     <model>
      <hist>
       <bucket>
         <freq> 10 </freq>
         <bv> Fiat Brava </bv>
       </bucket>
       <bucket>
         <freq> 15 </freq>
         <bv> Fiat Punto</bv>
       </bucket>
       <bucket>
         <freq> 8 </freq>
         <bv> Fiat Marea </bv>
       </bucket>
      </hist>
     </model>
     <city> Milan </city>
   </details>
 <details>
    <model>
      <hist>
        <bucket>
          <freq> 12 </freq>
           <bv> Fiat Brava </bv>
        </bucket>
<!-- other buckets are omitted for brevity -->
      </hist>
    </model>
    <city> Rome </city>
 </details>
</selling>
</car>
<car>
  <color> blue </color>
  <selling>
<!-- other histograms are omitted
for brevity -->
  </selling>
 </car>
</list>
```

Fig. 1.10. A sample XML synopsis document

1.5 Querying the XML Synopsis

XML Synopses have been proposed basically to answer aggregate queries in a fast and effective way and the language chosen to query XML data is the new standard proposal XQuery. Expressing aggregate queries in the relational data query language SQL is done by means of the GROUP-BY clause, but XQuery does not have such a powerful operator and aggregates are computed by (usually very complex) nested FOR clauses. The approach for the automatic translation of XQuery to the synopses considers users without any knowledge of the synopsis collection structure. In this case the translation derives directly from the original, complex and multi-nested query and cannot be optimized very well. Obviously, nobody denies to a more expert user to query directly the synopsis, writing the best query for the synopsis structure.

In the XQuery language, the aggregate functions available are COUNT, SUM, AVG, MAX and MIN. We develop our analysis considering the same set of functions and the presence of different kinds of data that, connected by a hierarchical structure, define different kinds of grouping. In the synopsis querying approach, we define new functions created to compute the aggregates over the histograms instead of the original data: these functions are `count_hist`, `sum_hist`, `avg_hist`, `max_hist` and `min_hist`. The structure of the query on the synopsis basically follows the structure of the original query; here are some general observations:

- in the construction of the synopsis, we have eliminated some elements from the document; therefore, we can find an answer only to queries involving elements that exist in the synopsis graph, otherwise we are forced to query the original data collection;

- since the construction of the synopsis does not change the path structure of the *crit_g* elements and stores all the values of their *active domain*, queries asking for values stored as grouping elements can find a precise answer (not approximate);
- queries involving histograms of categories (e.g., names or colors) always find a precise answer because each boundary value represents an exact value of the element *active domain* and the frequency of the buckets represents the exact count of this value in the target collection;
- the answer to a query involving histograms of numeric data (e.g., ages) is usually approximate, because the boundary values of the histograms are not exact values but intervals of values of the element *active domain*.

As an example, consider the synopsis shown in Fig. 1.10. In the synopsis, the *crit_g* element is the element *list/car/color*, which appears in the synopsis file with all its active domain values. In this case a query asking for the colors of the sold cars will find all the possible values of the element as if it were performed on the original data collection. Instead, consider the parameter document P constructed for the element price in Sect. 1.4: if we look for the number of cars that cost less than $12000, the answer on the synopsis will be approximate because it should use one of the buckets of the histogram of prices partially.

The rules that define the transformation QTR have been detailed in [Mar05]. As an example, consider the following rule:

QTR-Rule 1 *If the aggregate function in the original query is applied to a leaf element summ_e, the synopsis query uses the corresponding function on histograms (e.g.,* `count_hist`*,* `avg_hist`*).*

An example of the application of this rule is the following query, which asks for the number of Fiat Brava sold in Milan:

```
<total>
   for $det in doc("cars.xml")/list/car/selling/details
   where $det/city = "Milan"
   return count($det/model = "Fiat Brava"))
</total>
```

This query is transformed into:

```
<total>
for $det in doc("syn-cars.xml")/list/car/selling/details
where $det/city = "Milan"
return count_hist($det/model/hist/bucket/bv = "Fiat Brava"))
                          $QTR_Rule$
</total>
```

The other rules describe what appears in the clauses of the query, which function has to be used in case of categorical or numerical aggregate element,

and what happens in case of reverse steps in the query (see the details in [Mar05]).

1.6 Approximation analysis

In this section we further detail the considerations at the beginning of Sect. 1.5 about the approximation obtained in the synopsis querying process. Aim of this section is to analyze the approach in order to evaluate the degree of approximation during the querying phase.

We analyze our synopsis approach according to the following dimensions:

- *Coverage*: in this section we detail some general observations about the set of queries that can be answered using the synopses constructed with our methodology.
- *Answer quality*: the accuracy and confidence of its (approximate) answers to queries in $\{Q\}$.

Two further dimensions, the Footprint and the Query Time of the resulting synopsis, will be presented by examples in Section 1.7 during the analysis of the results obtained with the prototype tool.

1.6.1 Coverage and answer quality.

The structure of the query on the synopsis basically follows the structure of the original query, hence when querying the synopsis some general observations hold:

1. In the construction of the synopsis, we have eliminated some elements from the document; therefore, we can find an answer only to queries involving elements that exist in the synopsis graph, otherwise we are forced to query the original data collection.
2. Since the construction of the synopsis does not change the path structure of the *crit_g* elements and stores all the values of their *active domain*, queries asking for the existence of values stored as grouping elements can find a precise answer (not approximate); of course this statement does not hold if we are looking for aggregates (e.g., the number of...) involving *crit_g* elements.
3. Queries involving histograms of categories (e.g., names or colors) always find a precise answer because each boundary value represents an exact value of the element *active domain* and the frequency of the buckets represents the exact count of this value in the target collection.
4. The answer to a query involving histograms of numeric data (e.g., ages) is usually approximate, because the boundary values of the histograms are not exact values but intervals of values of the *active domain* of the element.

A histogram should reduce the data by describing the data distributions. For the sake of easiness, in this analysis we concentrate on leaf elements, but the considerations for non-leaf elements are similar but involve multidimensional histograms.

If histograms are available, we can use the synopsis to accelerate aggregate queries, using some ad-hoc formulas to build the aggregate functions `count_hist`, `sum_hist`, `avg_hist`, `max_hist`, and `min_hist`. For a detailed review of these formulas see [Mar05]. Moreover, in [Mar05] we consider the problem of refreshing the synopsis histograms when the target XML document collection is updated. Unfortunately, the W3C has not yet proposed an XQuery syntax for updates, although such extension is strongly needed. Only in the last few years the scientific community has started to formally deal with the problem of updates in XQuery and some preliminary works are [WR] or can be found in [Gal]. For this reason, we can only make some general considerations about the update of the synopsis but a deep study of this problem will be afforded as soon as XQuery will have a well defined update syntax.

1.7 Experimental results

A prototype tool (see [Mar05]) has been used to test the idea on a huge set of XML data in order to experimentally measure the approximation error and the benefits of using synopses in terms of computational costs and times. The prototype consists of two independent parts: *SynGenerator*, a tool for creating the synopsis and its DTD and *ApproXquery*, a tool for querying the synopsis. Both tools have been developed in Java. The first experiments allow us to make some preliminary observations: in our toy benchmark constructed by using the example DTD in Section 1.2.1 each car document file occupies about 500 bytes. If we construct the synopsis shown in Section 1.3, we see that it can occupy (in the worst case in which there is not more than one car model for each color and each town) about 750 bytes. Note that once a certain model-color-town group has been inserted in the synopsis, the dimensions of the synopsis itself do not change as the number of cars belonging to this group augments. If we consider a store able to sell 10 cars of a given color and model in a certain town, we have that the original document collection will occupy 5000 bytes while the synopsis still occupies 750 bytes, i.e., 15% of the original size. Obviously, the percentage space occupation decreases as the number of cars belonging to a certain group augments. Olap applications work with quantities of data enormously bigger with respect to our toy examples; hence it is easy to infer that synopsis collections can occupy an infinitesimal part of the original collection space, allowing a very good performance growth. In Figure 1.11 we show our car synopsis collection footprint and some comparative query execution times obtained by running some range queries presented in [Mar05]: the first column of the table shows the execution times of each original query on the target collection, the second column shows the execution times of the

Query execution times

Query	Query on the original collection	Query that creates the synopsis	Query on the synopsis
Q1	40.51 sec	29.33 sec	0.43 sec
Q2	51.10 sec	1 min 26 sec	0.65 sec
Q3	1 min 13 sec	49.58 sec	0.59 sec
Q4	1 min 02 sec	1 min 17 sec	0.61 sec

Footprint

	5 MB	/	17 KB (for all synopses)

(Intel Pentium 3, 1GHz, with 256 MB RAM)

Fig. 1.11. Comparative table of queries results

query to construct the synopsis, and the third column shows the execution times necessary to perform the transformed query on the synopsis. Even if the structures of the original and of the synopsis queries are very similar (they have been transformed automatically by the transformation QTR), the query execution times differ considerably. This is basically motivated by the dimensions of the data collections that have to be queried: indeed the synopsis has a footprint of 17 KB while the original collection occupies 5 MB.

1.8 Related work

In this section we present an overview of the main techniques used in literature to construct data synopses.

1.8.1 Synopsis data structures for relational data warehouses

A good starting point, for an overview on synopsis data structures for relational data warehouse, is [BDF+97], which describes the state of the art in *data reduction* techniques, for reducing massive data sets down to a "big picture" and for providing quick approximate answers to queries. The list of data structures that could be considered synopsis data structure is extensive. For example, Krishnan *et al* [KVI96] proposed and studied the use of a compact suffix tree-based structure for estimating the selectivity of an alphanumeric predicate with wildcards. Manber [man94] considered the use of concise *"signatures"* to find similarities among files. Broder *et al* [BCFM98] studied the

use of (approximate) min-wise independent families of permutations for signatures in a related context, namely, detecting and filtering near-duplicate documents. Other works include the use of multi-fractals and wavelets for synopsis data structures [FMS96, MVW98] and *join samples* for queries on the join of multiple sets [GPA+98].

For the purpose of this work we concentrate our review only on synopsis data structures used for fast approximate query answering.

In this setting, a good survey is [GM99], which describes a context for algorithmic work relevant to massive data and a framework for evaluating such work. Moreover the paper overviews the literature about synopses till 1999 and highlights results on some important problem domains from the database literature: frequency moments, hot list queries, and histograms and quantiles.

In the last 30 years, there have been a huge amount of works about synopses data structures applied in approximate answering approaches, whose main contributions are: 1) *histograms* [GMP97, GK01, PIHS96], that partition attribute values domain into a set of buckets; 2) *samples* [Olk93], which are based on the idea that a small random sample S of the data often well-represents the entire data; 3) *Wavelets* [VWI98], which are a mathematical tool for hierarchical decomposition of functions/ signals. Multi-dimensional data synopses are used to approximate the *joint data distribution* of multiple attributes [AGPR99]. They are used for the selectivity estimation for queries with multiple attributes and for approximating OLAP data cubes and general relations.

XML differs from relational data in several aspects:

- documents present a hierarchical structure, where the position of each element carries useful information;
- being semi-structured data, some items can be repeated or missing without a predefined document structure;
- there can be mixed content, numerical and categorical, stored in the same document.

The synopsis approach we have proposed in this work is constructed with the aim to save the information stored in the hierarchical structure of the XML document during the summarization process, to take into account the problems and the advantages given by the semi-structured nature of XML, and uses histograms, among all the proposed techniques in literature, because this technique seems the most suited to be used with the mixed content of XML.

1.8.2 XML data synopses

A first approach in the extension of synopsis approaches to XML is given by different techniques [FHR+02, CJF+01, PG02, AN03] that have recently been proposed for building statistics for XML data with the aim to estimate the

selectivity of path expressions; some example are: StatiX, an XML Schema-aware statistics framework that exploits the structure derived by regular expressions in the XML Schema in order to develop an efficient and accurate XML query result estimator; Chen et al. build statistics used to estimate the selectivity of tree pattern queries (also called *twig queries*), or branching path expressions; XSchetch, a synopsis-graph model addressing the optimization of XML queries posed over large volumes of XML data where the authors construct synopsis structures for enabling the estimation of the path and branching distribution in the data graph to be used for the optimization of the original query. [AN03] presents a technique for building on-line XML statistics by observing the XPath queries issued to data source and their result sizes. This technique stores the path expressions and the information about their selectivity for use in estimating the selectivity of future XPath queries. Instead, the approach we propose uses synopsis techniques to execute aggregate queries saving computational costs, paying, when necessary, a little loss in precision. To our knowledge, ours is the first work where synopses are used to find approximate answers to aggregate XML queries. The most recent, and to our knowledge unique, work for XML approximate query answers, at the time of this work, is [PGI04]: they study approximate query answers for XML queries focusing only on twig queries with branching path expressions, i.e., they consider the structural part of the problem and, w.r.t our work, ignore the value content of the document. Their approach is based on a structural XML synopsis, termed TREESKETCH, that captures, in limited space, the key properties of the underlying path distribution and enables approximate answers for one class of XML queries. Another difference between our work and the XML synopsis literature is that our synopses are constructed to answer a set of very frequent queries, using a methodology guided by the application.

1.9 Conclusions

In this work we have described the construction of a synopsis from an XML document collection and outlined the most important characteristics of the synopsis querying process. In this paper we suppose to store one synopsis graph ($summ\_e$ and corresponding $< crit\_g_1, \ldots, crit\_g_n >$) in each synopsis XML document, and use equi-width histograms as general purpose technique for summarizing XML data. In the next future we plan to focus on specific applications of the methodology in order to study other, more performative, statistical techniques. For instance, for collections of documents containing numerical data only, we could use the wavelets. Moreover we plan to extend our approach with a complete study of the problem of updating our synopses using XQuery, as soon as this language will have a standardized update syntax.

References

[AGPR99] Swarup Acharya, Phillip B. Gibbons, Viswanath Poosala, and Sridhar Ramaswamy. Join synopses for approximate query answering. pages 275–286, 1999.

[AN03] A. Aboulnaga and J. F. Naughton. Building xml statistics for the hidden web. In *Proc. CIKM'03 Conference*, New Orleans,Louisiana,USA, 2003.

[BCFM98] A. Z. Broder, M. Charikar, A. M. Frieze, and M. Mitzenmacher. Min-wise independent permutations. In *Proc. 30th ACM Symp. on the Theory of Computing*, pages 327–336, 1998.

[BDF+97] D. Barbarà, W. DuMouchel, C. Faloutsos, P. J. Haas, J. M. Hellerstein, Y. Ioannidis, H. V. Jagadish, T. Johnson, R. Ng, V. Poosala, K. A. Ross, and K. C. Sevcik. The new jersey data reduction report. In *Bulletin of Technical Committee on Data Engineering*, pages 20(4): 3–45, 1997.

[CJF+01] Z. Chen, H. V. Jagadish, F.Korn, N. Koudas, S. Muthukrishnan, R. T. Ng, and D. Srivastava. Counting twig matches in a tree. In *ICDE*, pages 595–604, 2001.

[FHR+02] J. Freie, J. R. Haritsa, M. Ramanath, P. Roy, and J. Simeon. Statix: Making xml count. In *ACM SIGMOD, Madison, Wisconsin, June 4-6*, 2002.

[FMS96] C. Faloutsos, Y. Matias, and A. Silberschatz. Modeling skewed distributions using multifractals and the '80-20' law. In *Proc. 22rd International Conf. on Very Large Data Bases*, pages 299–310, 1996.

[Gal] The galax project. http://www.galaxquery.org/.

[GK01] M. Greenwald and S. Khanna. Space-efficient online computation of quantile summaries. In *ACM Sigmod*, 2001.

[GM99] P. B. Gibbons and Y. Matias. Synopsis data structures for massive data sets. *DIMACS: Series in Discrete Mathematics and Theoretical Computer Science: Special Issue on External Memory Algorithms and Visualization*, vol. A, 1999.

[GMP97] P. B. Gibbons, Y. Matias, and V. Poosala. Fast incremental maintenance of approximate histograms. In *Proc. of Very Large Data Bases*, 1997.

[GMW99] R. Goldman, J. McHugh, and J. Widom. From semistructured data to xml: Migrating the lore data model and query language. In *Proc. WebDb*, pages 25–30, 1999.

[GPA+98] P. B. Gibbons, V. Poosala, S. Acharya, Y. Bartal, Y. Matias, S. Muthukrishnan, S. Ramaswamy, and T. Suel. Aqua: System and techniques for approximate query answering. In *Technical Report*, Murray Hill, New Jersey, 1998.

[HPG04] Jan Hidders, Jan Paredaens, and Dirk Van Gucht. A light but formal introduction to XQuery. In *Second International XML Database Symposium*, 2004.

[KVI96] P. Kishnan, J. S. Vitter, and B. Iyer. Estimating alphanumeric selectivity in the presence of wildcards. In *Proc. ACM SIGMOD International Conf. on Management of Data.*, pages 282–293, 1996.

[man94] U. manber. Finding similar files in a large file system. In *Proc. Usenix Winter 1994 Technical Conf.*, pages 1–10, 1994.

[Mar05] S. Marrara. *Aggregate queries in XQuery.* 2005. PhD Thesis, Politecnico di Milano, XVII PhD School Edition.

[MVW98] Y. Matias, J. S. Vitter, and M. Wang. Wavelet-based histograms for selectivity estimation. In *Proc. of ACM SIGMOD Conference*, pages 448–459, 1998.

[Olk93] F. Olken. Random sampling from databases., 1993. PhD Thesis, U.C. Berkeley.

[PG02] N. Polyzotis and M. Garofalakis. Statistical synopses for graph-structured xml databases. In *Proc. ACM SIGMOD Conference*, Madison,Wisconsin,USA, 2002.

[PGI04] N. Polyzotis, M. Garofalakis, and Y. Ioannidis. Approximate xml query answers. In *SIGMOD*, 2004.

[PIHS96] V. Poosala, Y. Ioannidis, P. Haas, and E. Shekita. Improved histograms for selectivity estimation of range predicates. In *Proc. ACM SIGMOD*, 1996.

[VWI98] J. S. Vitter, M. Wang, and B. Iyer. Data cube approximation and histograms via wavelets. In *Proc. the 7th Int. Conf. on Information and Knowledge Management.*, 1998.

[W3C99] W3C. Xml path language (XPath) version 1.0, 1999. http://www.w3.org/TR/xpath.

[W3C04] W3C. Xml query (XQuery) version 1.0, 2004. http://www.w3.org/XML/Query.

[WR] Ling Wang and Elke A. Rundensteiner. Updating xquery views.