Schema Summarization [*]

Cong Yu
Department of EECS
University of Michigan
congy@eecs.umich.edu

H. V. Jagadish
Department of EECS
University of Michigan
jag@eecs.umich.edu

ABSTRACT
Real database systems can often be very complex. A person wishing to access data from an unfamiliar database has the daunting task of understanding its schema before being able to pose a correct query against it. A schema summary can be of great help, providing a succinct overview of the entire schema, and making it possible to explore in depth only the relevant schema components.

In this paper we formally define a schema summary and two desirable properties (in addition to minimizing size) of a summary: presenting important schema elements and achieving broad information coverage. We develop algorithms that allow us to automatically generate schema summaries based on these two goals. We further develop an objective metric for assessing the quality of a schema summary using query information. Experimental evaluation using this metric demonstrates that the summaries produced by our algorithms can significantly reduce the amount of user effort required to formulate a query through schema exploration.

1. INTRODUCTION
Real databases often have extremely complex schemas. However, a complex schema is difficult to comprehend, limiting the database accessibility (in terms of both querying and data exchange) to a small number of people, who have spent a significant amount of time understanding the schema. Consider the example schema based on the XMark [12] benchmark in Figure 1. The schema is small compared to that of most production databases, and a significant portion of the schema has in fact been suppressed. Even so, a user unfamiliar with the XMark dataset will take time to figure out the major themes of the schema.

Typically, a user has a query need that depends on a portion of the schema. But to be able to express this query need, the user has to study the entire complex schema and discover the schema elements of interest. For example, a user who wishes to find out the end time for an open auction in the XMark database, has to study the schema and filter away irrelevant information about items and persons. These problems become much worse for more complex schemas, especially when the schema can no longer be presented to the user in its entirety at a reasonable information density [15].

[*]Supported in part by NSF under grant IIS-0438909, and NIH under grant 1-U54-DA021519.

In this paper, we propose the notion of *schema summary* to address the above problems. As shown in Figure 2(A), a summary utilizes *abstract elements* and *abstract links* to summarize a complex schema and provide the users with a concise overview for better understanding. Each abstract element in the summary corresponds to a cluster of original schema elements (and other lower level abstract elements in the case of a multi-level summary), and each abstract link represents one or more links between the schema elements within those abstract elements. A user presented with the summary in Figure 2(A) can immediately understand that the schema is about auctions, along with the items and persons related to the auctions. Furthermore, if the user is interested in only information about open auctions, she can selectively expand that abstract element, as shown in Figure 2(C). She will then get more detailed information for that particular part of the schema alone, without being exposed to other unrelated details.

While schema summaries are useful, creating a good summary is a non-trivial task. A schema summary should be concise enough for users to comprehend, yet it needs to convey enough information for users to obtain a decent understanding of the underlying schema and data. Consider the two schema summaries in Figure 2(A & B). While both have the same number of abstract elements, A is intuitively a better summary than B because A informs the user about the `bidder` element in the schema, which corresponds to much more information (i.e., the bidders) in the database than the `region` element in B. In this paper, we capture this intuition formally through the notions of *summary importance* and *summary coverage*. Along with the intuitive notion of *summary size*, they describe the effectiveness of a summary for a complex schema and the database associated with it.

Humans can be good at summarization, and the database designer could generate a schema summary at the time the schema is being specified. In fact, a design summary (usually expressed in ER diagram) is sometimes created as part of the design process. However, such internal documents are unlikely to be made available, in a heterogeneous environment, to external users who may be permitted database access. On the other hand, it is exactly such users who would benefit most from having a schema summary available. Therefore, we have little choice but to generate summaries from existing databases. One possible approach is to generate summaries manually, but this is labor-intensive, and is impractical in

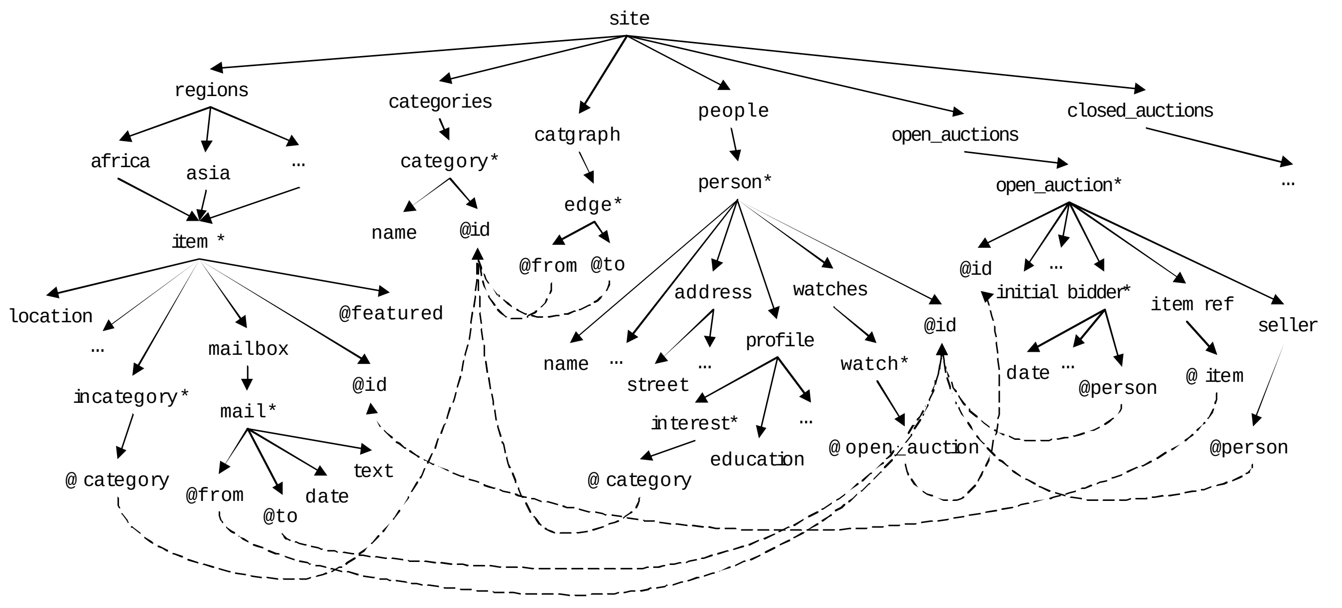

Figure 1: Example schema based on XMark. Nodes, solid arrows, and dashed arrows represent schema elements (or attributes, with prefix '@'), structural links, and value links, respectively. Elements with suffix '*' are of SetOf type.

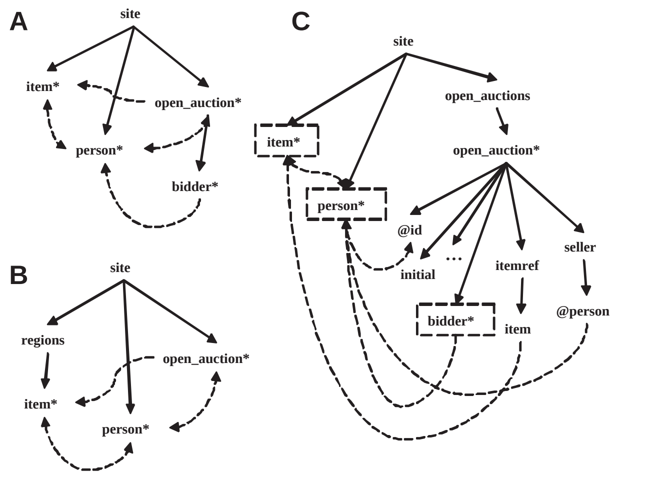

Figure 2: Two full schema summaries (A & B) and an expanded schema summary (C) for the XMark schema in Figure 1. All elements in A & B are abstract elements (except site), while dashed boxes indicate abstract elements in C. Dashed arrows indicate value links or abstract links representing at least one value link.

situations where schemas evolve. Leaving summary generation to a manual process also leaves open the possibility that the summary will not be updated when the schema (and/or the database) is updated, resulting in a supposed summary that is actually outdated and misleading. In short, we would like to have automatic summary generation. In this paper, we design and implement a system that can accomplish this for schemas in both relational and hierarchical data models.

While summarizing schema, whether relational or hierarchical, has never been studied before, several studies have focused on ER model abstraction [13, 1, 3, 4, 11]. Those methods aim to cluster ER entities into abstract entities and rely heavily on the semantic types of the relationships between entities. For example, two entities linked with an is-a relationship are typically grouped together. Unlike ER models, however, relational or hierarchical schemas do not have semantic meanings attached to the structural or value links. Therefore, the above methods can not be applied to schema summarization. Furthermore, the ER model abstraction can not take advantage of semantic information embedded in the data since it only considers the structural information inside the ER model. Our notions of summary importance and coverage address these problems by taking into consideration both schema structure and data distribution for generating effective summaries automatically.

Main contributions and paper outline: We make the following main contributions: I. We formally define the notion of *schema summary* (Section 2); II. We introduce *summary importance* and *summary coverage* as desirable properties for a good schema summary (Section 3); III. We design novel algorithms for automatic generation of summaries that balance importance and coverage (Section 4); IV. We propose a query-based metric for objectively evaluating the quality of schema summary and demonstrate, through a comprehensive set of experiments, that automatic summaries generated by our system significantly reduce user effort required in formulating a query (Section 5).

2. SCHEMA SUMMARY

We consider *schemas* as labeled directed graphs (Figure 1), which model both relational and hierarchical (XML) schemas. Each node in the graph corresponds to: 1) a relation or an attribute column for relational schemas; 2) an element or an attribute for hierarchical schemas. The edges correspond to structural links (e.g., parent-child links) and value links (e.g., foreign key constraints). A special node is designated as "root." It has no incoming structural link, and corresponds to the root of a hierarchical schema. For relational schemas, we introduce an artificial root node with outgoing structural links to all relation nodes. Formally, we have:

Definition 1 (Schema). *A* schema *is defined to be a labeled directed graph, $SG = \langle \mathcal{E}, \mathcal{S}, \mathcal{V}, r\rangle$, where:*

– $\mathcal{E}$ is a finite set of **elements***. Each $e \in \mathcal{E}$ has a label l and a type τ, where τ is a regular expression: τ ::=* SetOf τ | Simple | (Rcd | Choice)$[e_1 : \tau_1, ..., e_n : \tau_n]$;

– $\mathcal{S}$ is a finite set of **structural links** *between elements. Each link $(e_1 \rightarrow_S e_2) \in \mathcal{S}$, $e_1, e_2 \in \mathcal{E}$, is a result of e_1 having an associated type of τ or* SetOf τ*, where τ ::=* (Rcd|Choice)$[..., e_2 : \tau', ...]$*, e_1 is called* **parent element** *and e_2 is called* **child element***;*

– $\mathcal{V}$ is a finite set of **value links** *between elements. Each link $(e_1 \rightarrow_V e_2) \in \mathcal{V}$, $e_1, e_2 \in \mathcal{E}$, is a result of an inclusion constraint $\tau_1[e'_1] \subseteq \tau_2[e'_2]$, where τ_1 ::=* (Rcd|Choice)$[..., e'_1 : \tau'_1, ...]$ *is a type associated with e_1, τ_2 ::=* (Rcd|Choice)$[..., e'_2 : \tau'_2, ...]$ *is a type associated with e_2, and τ'_1, τ'_2 ::=* Simple | SetOf Simple*, e_1 is called* **referrer element** *and e_2 is called* **referee element***;*

– $r \in \mathcal{E}$, called the root element, is the only element with no incoming structural link.

Type Simple is the atomic value type (e.g., str, int, etc.) representing relational columns, XML attributes, and XML elements with atomic values. Furthermore, in relational schemas, elements with SetOf Rcd type are used to represent relations. And in hierarchical schemas, elements with SetOf type represent schema elements with `maxOccurs` greater than one, while elements with Rcd and Choice types represent the "all" and "choice" model-groups, respectively. We ignore order and represent the "sequence" model-group as elements with Rcd type. Furthermore, for n-ary value links, we simply decompose them into multiple unary value links. Extensions that consider element order and n-ary value links are not conceptually difficult, but are notationally cumbersome and hence not presented here. Finally, while value links in Figure 1 always connect two simple elements, they are considered as links between the two parent elements containing those simple elements. This is necessary because, semantically, the connection is between the two parents, and the simple elements are introduced because of syntactic necessity. For example, the semantics of the value link between `bidder/@person` and `person/@id` in Figure 1 is that each bidding action is performed by a person: a connection between `bidder` and `person`.[1] Extending the definition of schema, we have:

Definition 2 (Schema Summary). *A* summary *of schema $SG = \langle \mathcal{E}, \mathcal{S}, \mathcal{V}, r\rangle$ is defined to be a labeled directed graph, $SS = \langle \mathcal{E}', \mathcal{S}', \mathcal{V}', \mathcal{M}, \mathcal{AE}, \mathcal{AL}, r'\rangle$, where:*

– $r' = r$;

– $\mathcal{E}' \subseteq \mathcal{E}$, $\mathcal{S}' \subseteq \mathcal{S}$, $\mathcal{V}' \subseteq \mathcal{V}$;

– $\mathcal{M}$ is a finite set of correspondences between original schema elements and abstract elements, each $(e \sqsubset e') \in \mathcal{M}$, where $e \in \mathcal{E}$ and $e' \in \mathcal{AE}$, means that e is directly or indirectly represented by e' in the summary;

– $\mathcal{AE}$ is a finite set of abstract elements;

[1]Note that, semantically, the `bidder` element represents the bidding action, and the actual information about the bidder itself is stored at the `person` element, which is connected via a value link.

– Each $e \in \mathcal{E}' \cup \mathcal{AE}$ has a label l and a type τ, where τ is a regular expression: τ ::= Abstract τ | SetOf τ | Simple | (Rcd | Choice)$[e_1 : \tau_1, ..., e_n : \tau_n]$*, and each $e \in \mathcal{AE}$ has an associated type* Abstract τ *or* SetOf Abstract τ*.*

– For each $e \in \mathcal{E}$: 1) $e \in \mathcal{E}'$, or 2) there exists $e' \in \mathcal{AE}$, s.t. $(e \sqsubset e') \in \mathcal{M}$. Similarly, for each $e' \in \mathcal{AE}$, there exists $e \in \mathcal{E}$ s.t. $(e \sqsubset e') \in \mathcal{M}$;

– $\mathcal{AL}$ is a finite set of abstract links;

– For each $l_s = (e_1 \rightarrow_S e_2), l_s \in \mathcal{S}$: 1) $l_s \in \mathcal{S}'$, or 2) there exists $(e'_1 \rightarrow_{AL} e'_2) \in \mathcal{AL}$, where $(e_1 \sqsubset e'_1) \in \mathcal{M}$ (or $e_1 = e'_1 \in \mathcal{E}'$) and $(e_2 \sqsubset e'_2) \in \mathcal{M}$ (or $e_2 = e'_2 \in \mathcal{E}'$), or 3) e_1 and e_2 are represented by the same $e \in \mathcal{AE}$;

– For each $l_v = (e_1 \rightarrow_V e_2), l_v \in \mathcal{V}$: 1) $l_v \in \mathcal{V}'$, or 2) there exists $(e'_1 \rightarrow_{AL} e'_2) \in \mathcal{AL}$, where $(e_1 \sqsubseteq e'_1) \in \mathcal{M}$ (or $e_1 = e'_1 \in \mathcal{E}'$) and $(e_2 \sqsubseteq e'_2) \in \mathcal{M}$ (or $e_2 = e'_2 \in \mathcal{E}'$), or 3) e_1 and e_2 are represented by the same $e \in \mathcal{AE}$.

Intuitively, each abstract element in a schema summary represents a group of original schema elements and a single element is chosen as the representative of each group. The abstract element assumes the identity of the representative element and the remaining elements are hidden from view. Links between elements within the group are also hidden, while links connecting elements inside the group with outside elements are consolidated and represented as abstract links. An original schema element is *directly represented* by an abstract element if it is chosen as the representative element (e.g., element `person` for the abstract `person` element in Figure 2(A)). Otherwise, it is *indirectly represented* (e.g., element `profile`, which is indirectly represented by the abstract `person` element). The goal of schema summarization is to select appropriate schema elements as the set of abstract elements, and to determine which of the remaining schema elements each abstract element represents. Elements in a summary are also called *summary elements*, regardless whether they are abstract or not.

A schema summary is a *full summary* (e.g., Figure 2(A)) if it contains only abstract elements (except root); otherwise, it is an *expanded summary* (e.g., Figure 2(C)). An expanded summary can be especially helpful for users seeking detailed information within a confined component of the schema. In addition, an abstract element can itself be represented by another abstract element, thus creating a *multi-level summary*, which can be helpful for a user facing extremely large schemas. We note here that the original schema itself can be regarded as a fully expanded summary with empty $\mathcal{M}, \mathcal{AE}, \mathcal{AL}$. Similarly, a full summary can be considered to have an empty $\mathcal{E}'$ (except root), $\mathcal{S}', \mathcal{V}'$. In this paper, we focus on automatic generation and objective evaluation of full summaries and believe that the same techniques can be applied to expanded and multi-level summaries with relatively simple extensions.

3. SCHEMA SUMMARY QUALITY

Having defined a schema summary formally, we can now turn to the discussion of summary properties that influence its quality. The first obvious property is the *summary complexity*, which can be defined as the number of elements in the summary (i.e., summary size). The more elements a summary contains, the more complex it is for the user to comprehend. Therefore, all other properties being equal, a smaller summary is always preferred to a larger summary. In addition to summary complexity, we would like a summary to be informative, in that a user, looking at the summary, should get a good idea of the overall structure of the schema. What does "good idea" mean? There are a few desiderata: we should select elements that are "important," or in other words, more representative of the schema, in preference to nodes that are not; and we should select elements that are broadly distributed throughout the schema: a collection of elements in one local component of the schema is not likely to be a good summary for all the users. Based on these desiderata, we define two main properties in the next two subsections: *Summary Importance* and *Summary Coverage*.

3.1 Summary Importance

Not all schema elements are created equal. Consider the schema element `person` in the XMark schema in Figure 1 as an example. Most people would agree that `person` is more important than both element `watch` and element `people`. Trying to advance an objective reason for this intuition, we notice that `person` is much better connected in the schema than the element `watch`, and it is also higher up in the hierarchy. Meanwhile, `people` is even higher in the hierarchy than `person`, but in a typical database we expect few instances of the former and many instances of the latter, making it more likely that `person` rather than `people` will be the query focus.

In other words, the importance of a schema element is reflected in two aspects – its connectivity in the schema and its cardinality in the database. The connectivity of an element in the schema graph provides a count of the number of other elements that are directly connected to it (via either structural or value links). An important element is likely to be one from which many other elements can be reached easily. The cardinality of a schema element is the number of data nodes it corresponds to. If there are many data nodes of a schema element in the database, then that element is likely to be of greater importance than another one with very few data nodes. We note that, unlike connectivity, cardinality is determined by the data distribution and not by the schema structure.

As we attempt to develop a single comprehensive notion of importance that combines these two aspects, we notice some similarities to the web. The importance of a web page is determined by both the goodness of the match of the search terms on the page itself ("cardinality" of search terms on the page) and the links connecting to it from other important pages ("connectivity" of the page) [2]. Adapting this idea to our context, we can similarly define:

FORMULA 1 (SCHEMA ELEMENT IMPORTANCE). *The* importance *of a schema element e, written as I_e, w.r.t. a schema and a database conforming*[2] *to the schema, depends on its connectivity in the schema and its cardinality in the database, and can be calculated with the following iterative formula until convergence is reached:*

$$I_e^r \ = \ p * I_e^{r-1} + (1-p) * \sum_j W_{e_j \to e} * I_{e_j}^{r-1}$$

where $W_{e_j \to e} = \frac{RC(e_j \to e)}{\sum_k RC(e_j \to e_k)}$.

W is the **neighbor weight***, which is the relative weight of an element (e) from an element (e_j) it directly connects, compared with all other elements (e_k) directly connected to the latter; r denotes the number of iterations; j ranges over all the schema elements connected to e; for each such element e_j, k ranges over all the schema elements connected to e_j (including e itself); $RC(e_j \to e_k)$ calculates the* **relative cardinality***, i.e., the average number of e_k data nodes connected to each e_j data node; and finally, $0 \le p \le 1$ is a tuning parameter we call* **neighborhood factor***. The lower the p, the more the importance of an element is affected by the elements it is connected to. For all elements, the initial importance I_e^0 is set to the cardinality of the element in the database.* $\square$

There are two things worth mentioning here. First, the relative cardinality $RC(e_1 \to e_2)$ is calculated from the database as the average number of e_2 data nodes connected to each e_1 data node[3]. For example, the relative cardinality from `open_auction` to `bidder` can be 5 (i.e., on average 5 bidding actions per auction), and the relative cardinality from `bidder` to `open_auction` is 1 (a bidding action is always tied to one auction). Second, the sum of the importance values of all schema elements remains unchanged from iteration to iteration: it is simply the sum of the cardinalities of all schema elements in the database.

Intuitively, an element will have a high importance value if: 1) it connects to many other elements (because more elements will contribute their importance values to it), especially important elements; 2) it has a high relative cardinality from another element, in comparison with other elements connected to that element, because a majority of the importance from the latter will be transferred to it due to the high neighbor weight. Considering the XMark schema being used as our running example, the most important elements are `bidder`, `item`, and `person`, in that order, with importance scores of 190292, 143881, and 128465, respectively (calculated with a dataset generated using scale factor of 1).

Based on schema element importance, the *summary importance* can be defined as:

DEFINITION 3 (SUMMARY IMPORTANCE). *The* importance *of a schema summary SS w.r.t. a schema SG and a database conforming to the schema, written as R_{SS}, is defined as the ratio between the total importance of (abstract and non-abstract) elements in the summary versus the total element importance in the original schema:*

$$R_{SS} = \frac{\Sigma_i(I_{e_i} \mid e_i \in SS.\mathcal{E}' \cup SS.\mathcal{AE})}{\Sigma_j(I_{e_j} \mid e_j \in SG.\mathcal{E})}$$

A very small summary should contain only important elements and should still have an importance value that is significant. As the size of the summary increases, we expect its importance value to increase as well, rapidly at first, but then slower and slower, until it asymptotically reaches a value of 1 when the summary size becomes equal to the original schema size. Note that the denominator of the fraction for importance is really the sum of all element cardinalities in the database.

[2]We adopt the notion of conformance defined in [16].

[3]While sometimes the schema can provide this information, in general, it can only be derived from the database itself.

3.2 Summary Coverage

Intuitively, a summary is not good if all the summary elements are immediate neighbors in a high cardinality portion of the schema graph, leaving the rest of the schema very poorly covered in the summary. Consider the schema elements `address` and `interest` (both are descendants of `person`) in the running example in Figure 1. Due to its high connectivity (it has a total of 5 child elements, only one is shown in the figure for simplicity), element `address` may gather enough importance value to make it more important than `interest`. However, if element `person` is already selected into the summary, then `address` is a less desirable candidate to be selected into the summary than `interest` because of the closeness between `person` and `address`. This closeness is to a certain extent reflected in the fact that each `person` has only one `address`, implying that `address` is semantically attached to `person`, and therefore is "covered" by `person` in the summary.

While summary importance measures how much a summary captures important schema elements, it clearly does not capture how well the summary covers the overall schema and database content. To understand this notion of coverage, we first look at the closeness between schema elements.

The most straightforward metric for closeness between two nodes in a graph is the minimum number of (structural and value) links it takes (i.e., the steps) to traverse from one to the other. This, however, is an ineffective metric for measuring the closeness between schema elements. Consider again the running example, and the elements `bidder` and `seller`. While both are one step away from `open_auction`, semantically, `bidder` is further away from `open_auction` than `seller`, because each open auction has exactly one seller but multiple bidders. Based on this observation, we argue that the closeness between two schema elements is a combination of both the number of steps between the two and the relative cardinality along each of those steps. Formally, we have:

FORMULA 2 (SCHEMA ELEMENT AFFINITY). *The* affinity *of schema element e_a to schema element e_b, written as $A_{e_a \to e_b}$, w.r.t. a schema and a database conforming to the schema, can be calculated with the following formula:*

$$A_{e_a \to e_b} = \max_i (A^{path_i}_{e_a \to e_b})$$

$$A^{path_i}_{e_a \to e_b} = \frac{1}{n_i} \prod_{j=2}^{n_i} \frac{1}{RC(e_{j-1} \to e_j)}$$

where i ranges over all possible paths between e_a and e_b, for each $path_i$ of length n_i, j ranges over all elements along the path, $e_1 = e_a$ and $e_{n_i} = e_b$. For the special case where $e_a = e_b$, $A_{e_a \to e_a} = 1$. □

If there are multiple paths from one element to the other, only the path resulting in the highest affinity is considered. The affinity along any path is obtained as the product of the edge affinities, but is also divide by the length of the path. This division is necessary to penalize longer paths – even if each individual edge along a path has affinity 1, we intuitively would not want the affinity between the elements at the end of a long chain still to be 1.

Also, note that schema element affinity is directed, and $A_{e_a \to e_b}$ is often different from $A_{e_b \to e_a}$. This is not surprising since the affinity of a child element toward a parent element is intuitively different from the affinity of the parent toward the child: each child has only one parent while each parent may have multiple children and therefore be less close to any single one of them.

With the notion of affinity, two things are now possible. First, given a set of schema elements selected into the summary (to become summary elements), each remaining schema element can be assigned to the summary element toward which it has the highest affinity. This produces element groups, specifying which schema elements are represented by which summary element. Second, each summary element can now be considered as covering the elements that it represents, with the amount of coverage determined by the affinity it has toward the element being covered:

FORMULA 3 (SUMMARY ELEMENT COVERAGE). *The* coverage *of schema element e_b by schema element e_a, written as $C_{e_a \to e_b}$, w.r.t. a schema and a database conforming to the schema, can be calculated with the following formula:*

$$C_{e_a \to e_b} = Card_{e_b} * \max_i (C^{path_i}_{e_a \to e_b})$$

$$C^{path_i}_{e_a \to e_b} = \prod_{j=2}^{n_i} (A_{e_{j-1} \to e_j} * W_{e_j \to e_{j-1}})$$

where i ranges over all possible paths between e_a and e_b, for each $path_i$ of length n_i, j ranges over all elements along the path, $e_1 = e_a$ and $e_{n_i} = e_b$. For the special case where $e_a = e_b$, $C_{e_a \to e_a} = Card_{e_a}$. □

We could have used affinity for coverage directly, but good affinity toward an element does not guarantee good coverage, as we shall see in the example below. Instead, we have chosen to weight the affinity of each edge along a coverage path with its neighbor weight (W) as defined in Formula 1. The neighbor weight from e_a to e_b essentially captures the relative amount of information flowing from e_a that is distributed to e_b, among all the elements that are connected to e_a. Intuitively, affinity measures the internal ability an element has to cover the other element, while neighbor weight measures the competition an element faces in covering the other element.

Consider the elements `(b)idder` and `(o)pen_auction` in the running example in Figure 1. Assume the $RC(\texttt{b} \to \texttt{o})$ and $RC(\texttt{o} \to \texttt{b})$ are 1 and 2, respectively, and `o` has, in addition to `b`, a total of 10 children, each with a relative cardinality of 1. The affinities $A_{\texttt{b} \to \texttt{o}}$ and $A_{\texttt{o} \to \texttt{b}}$ will be close to 1.0 and 0.5, respectively. Since they are directly connected, the coverage can simply be calculated as the product of the cardinality, the affinity, and the neighbor weight. The coverage $C_{\texttt{o} \to \texttt{b}}$ can therefore be calculated as $Card_{\texttt{b}} * 0.5 * \frac{1}{1} = 0.5 * Card_{\texttt{b}}$. Here, the affinity is multiplied by the factor $\frac{1}{1}$: `o` is the only parent element that `b` has. On the other hand, the coverage $C_{\texttt{b} \to \texttt{o}}$ is calculated as $Card_{\texttt{o}} * 1.0 * \frac{2}{10+2} = 0.17 * Card_{\texttt{o}}$. Here, the affinity is multiplied by the factor $\frac{2}{10+2}$: `b` is only one of the many child elements that `o` has. As we can see from this example, while coverage is closely related to affinity, the higher affinity that `b` has for `o` does not result in greater coverage. This is not surprising, especially for child elements, because even though they are very close to their parents, the information content at their parents are typically beyond their scope.

With schema element affinity and coverage, we can now formally define *summary coverage*:

Definition 4 (Summary Coverage). *The* coverage *of a schema summary SS, written as C_{SS}, w.r.t. to a schema SG and a database conforming to the schema, is the ratio between the total coverage of all schema elements by elements in the summary and the total coverage of all schema elements by the original schema (which is the same as the total cardinality of all schema elements):*

$$C_{SS} = \frac{\sum_i \sum_j (C_{e_i \to e_j} \mid (e_j \sqsubset e_i) \in SS.\mathcal{M} \text{ or } e_i = e_j \in SS.\mathcal{E}')}{\sum_k (C_{e_k \to e_k} \mid e_k \in SG.\mathcal{E})}$$

Intuitively, summary coverage reflects how well all the summary elements collectively cover the schema being summarized, with each individual summary element responsible for the schema elements it represents. In the original schema, which can be viewed as a fully expanded summary, each element is represented by itself only, and the total coverage is equal to the total cardinality of all the schema elements (the denominator).

3.3 Discussion

To summarize, we consider two properties of goodness for a summary: importance and coverage. These two properties are somewhat correlated, but are inherently different, and it is certainly possible to find two summaries where one has more importance but the other has better coverage: an ideal summary should balance these two properties. In our system, we achieve the balance by selecting important elements that do not substantially overlap with another element already in the summary.

Using the database content (i.e., data distribution) as one of the criteria guiding summary generation has never been attempted before, both because previous studies have focused on ER model abstraction where data is not available, and because of traditional boundaries separating work focused on schema and work focused on data. We show here that while schema structure is of vital importance in summarization, data distribution often provides important insights that significantly improve the summary quality. One consequence of using data distributions is that the generated summary may evolve when the database is updated even though the schema stays the same. We view this as an advantage instead of an disadvantage. If the summary is generated based on a database of sufficient size, minor changes to the database will not affect the summary. In fact, if the changes follow the same data distribution as the old database, the summary will not be affected even when the changes are major. When the data distribution has changed significantly because of changes to the database, we argue that the focus of the database has been shifted, and as a result, a change in the summary is indeed appropriate.

4. EFFICIENT SCHEMA SUMMARIZATION

Based on the formulas presented in Section 3, we design algorithms to automatically generate schema summaries given a target summary size. The key issue here is how to select schema elements to be included in the schema summary as the representatives of the abstract groups. Algorithm *MaxImportance* (Section 4.2) iteratively computes the importance values for all schema elements until convergence and selects elements for a schema summary with the maximum summary importance. Algorithm *MaxCoverage* (Section 4.3) adopts a dominance-based pruning strategy to select elements for a schema summary with the highest summary coverage. Finally, in Section 4.4, we provide a heuristic for combining these two and select elements for a schema summary that balances importance and coverage.

```
Function annotateSchema($SG$, $D$):
 1. foreach element $e \in SG.\mathcal{E}$:
 2.   $e.Card = 0$;
 3.   foreach (structural or value) link $l$ of $e$: $e.l.Card = 0$;
 4. Visit $D$ in a depth-first preorder traversal using a stack;
 5. let $n \in D$ be the data node currently being visited:
 6.   let $e \in SG.\mathcal{E}$ be the schema element for $n$;
 7.   let $e' \in SG.\mathcal{E}$ be the parent element for $e$;
 8.   let $e'' \in SG.\mathcal{E}$ be the referee element for $e$;
 9.   $e.Card{+}{+}$;
10.   let links $e.l_1, e'.l_2 = (e' \to_S e)$:
11.       $e.l_1.Card{+}{+}$; $e'.l_2.Card{+}{+}$;
12.   let links $e.l_3, e''.l_4 = (e \to_V e'')$:
13.       $e.l_3.Card{+}{+}$; $e''.l_4.Card{+}{+}$;
14. foreach link $l = (e_1 \to e_2) \in SG.\mathcal{S} \cup SG.\mathcal{V}$:
15.   $RC(e_1 \to e_2) = \frac{e_1.l.Card}{e_1.Card}$; $RC(e_2 \to e_1) = \frac{e_2.l.Card}{e_2.Card}$;
16. return $SG$
```

Figure 3: Function annotateSchema: SG is the schema graph to be annotated and D is the database conforming to SG.

Given the set of selected schema elements, which serve as the abstract elements in the summary, generating schema summary is simply assigning each remaining schema element to its closest abstract element and establishing abstract links between those elements. The algorithm is straightforward and not presented due to space limitations.

4.1 Annotating Schema Graph

Before we introduce the algorithms, we first need to annotate the schema graph with relative cardinalities required in all the formulas. As shown in Figure 3, we perform a depth-first traversal of the database. During the visit of each node, we update the cardinality of its corresponding schema element and the cardinality of each link connecting that element with its *parent* or *referee* element. Notice that the cardinality of a link pointing to the *child* or *referrer* element is updated only when the nodes corresponding to the *child* or *referrer* elements are visited. Finally, the relative cardinalities are calculated based on element cardinalities and link cardinalities.

4.2 Maximizing Summary Importance

There are two parts to computing a schema summary with highest summary importance. The first part is to compute the importance values of each schema element. The second part is to choose the K most important elements to obtain a summary of size K, which just requires a simple (partial) sort, and is trivial. The first part involves initializing the importance of each schema element to its cardinality in the data set and then iteratively applying Formula 1 until the importance values converge (i.e., for each element, the difference between the old and the new importance value is less

```
Algorithm MaxImportance:
Input: Schema $SG$, Database $D$, summary size $K$
 1. $SG$ = annotateSchema($SG$, $D$);
   // $SG$ is represented as an array of elements
 2. foreach element $e \in SG.\mathcal{E}$:
 3.   $I_e^{cur} = e.Card$;
 4. done = false;
 5. while (!done): // a limit on the # iterations can also be set
 6.   done = true;
 7.   foreach element $e \in SG.\mathcal{E}$:
 8.     calculate $I_e^{new}$ using Formula 1;
 9.     if $\frac{|I_e^{new} - I_e^{cur}|}{I_e^{cur}} > c$: done = false; // typically, c = 0.1%
10.     $I_e^{cur} = I_e^{new}$;
11. Sort elements according to their importance values;
Output: $E$, the set of $K$ most important elements
```

Figure 4: Algorithm MaxImportance.

than some threshold). As shown in Figure 4, we represent the schema graph as an array of elements, each with an array of links pointing to the elements connected directly to it. At each iteration, the algorithm scans through the graph and computes the new importance values.

Convergence and Complexity: The proof of convergence is similar to the one used in [8]. For the schemas we have tested, the algorithm typically converges within several hundred iterations for $p = 0.5$ and threshold $c = 0.1\%$. At each iteration, the algorithm essentially visits each link twice (once for each end), and therefore has a time complexity of $O(N^2)$ per iteration, where N is the number of elements in the schema. However, the worst case complexity is only encountered when the schema is almost fully connected: for typical schemas with an average fanout of f, the average complexity is $O(fN)$. The complexity of the second part is no worse than $O(N\log N)$ and we can typically do better since we only need the top K values.

4.3 Maximizing Summary Coverage

Generating a schema summary of K elements with the highest summary coverage, on the other hand, is more complicated. Simply selecting the top K schema elements with the highest element coverage for other elements, and putting them together, does not guarantee us a summary with the highest summary coverage. This is because two high coverage elements may have a significant overlap in their coverage, producing a summary coverage[4] much smaller than the sum of individual element coverages. An optimal strategy therefore must generate candidate sets of K elements from a total of N elements in the schema, compute the summary coverage for each set, and select the set with the highest summary coverage. The number of sets to be examined is $O(\mathcal{C}_N^K)$, which is approximately $O(N^K)$ for $K \ll N$. For each candidate set, a two step computation is required. First, the affinity is computed from each unselected element to each selected element. Based on this, we can assign each unselected element to a selected element to which the former has greatest affinity. Second, the coverage is computed for each selected element, with regard to each unselected element that is assigned to it. Together, they result in $O(KN^2)$ time complexity[5]. Therefore, the overall complexity of this simple algorithm is $O(KN^2\mathcal{C}_N^K)$. For complex schemas with large N, this naive algorithm becomes impractical, and we design a novel pruning strategy to reduce the time cost.

[4]Recall from Definition 4 that in calculating the summary coverage, an element in the original schema can only be covered by exactly one summary element, toward which it has the highest affinity.

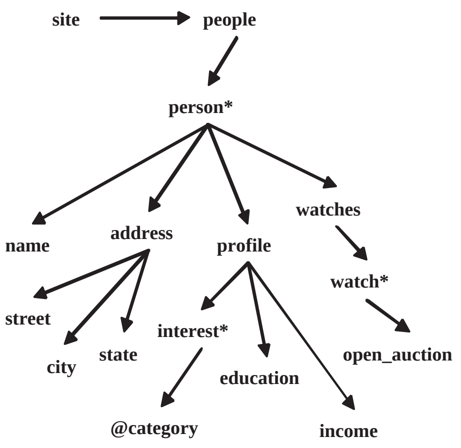

Figure 5: Partial schema extracted from Figure 1.

Candidate Set Pruning: It turns out that the number of candidate sets can be greatly reduced because certain schema elements are *dominated* by other elements in terms of coverage, and therefore are less worthy of appearance in a summary. We define element e_A *dominates* element e_B to be the following: for any summary SS, if only e_B is in SS, then we can always replace e_B with e_A and obtain a new summary SS', which has a higher summary coverage than SS. The dominance relationship can be determined according to Theorem 1:

THEOREM 1. *For any two schema elements e_1 and e_2. Let E be the set of schema elements (including e_2) with higher coverage by e_2 than by e_1, e_c be the schema element, besides e_1 itself, with the highest coverage of e_1, $C_1 = \sum_j(C_{e_1 \to e_j} \mid e_j \in E)$, $C_2 = \sum_j(C_{e_2 \to e_j} \mid e_j \in E)$, e_1 dominates e_2 in terms of coverage if:*
- $C_2 - C_1 \le Card_{e_1} - C_{e_2 \to e_1}$, and
- $C_2 - C_1 \le Card_{e_1} - C_{e_c \to e_1}$ if $e_c \neq e_2$.

Proof: Consider two otherwise identical summaries, SS_2 containing e_2 and SS_1 replacing e_2 with e_1. By choosing SS_1 over SS_2, we lose coverage for elements in E, but also gain coverage for at least e_1. The difference $C_2 - C_1$ represents the maximum decrease in coverage if we choose SS_1 over SS_2. If $e_c = e_2$, then the minimum increase in summary coverage is $Card_{e_1} - C_{e_2 \to e_1}$. If $e_c \neq e_2$, then $Card_{e_1} - C_{e_c \to e_1}$ represents the minimum summary coverage increase if e_c is in the summary. Satisfying the two conditions guarantees that the minimum coverage increase is greater than the maximum coverage decrease, therefore SS_1 has a higher coverage than SS_2. □

Consider the schema shown in Figure 5, which is a partial extraction of the running example schema. We assume all relative cardinalities are 1, except for three links: RC(`profile` $\to$ `interest`), RC(`watches` $\to$ `watch`), and RC(`people` $\to$ `person`), which are greater than 1 (meaning

[5]We ignore here the path length l and number of alternative paths n between elements for the simplicity of analysis: if considered, the complexity is $O(l^2n^2KN^2)$.

```
Algorithm MaxCoverage:
Input: Schema $SG$, Database $D$, summary size $K$
 1. $SG$ = annotateSchema($SG$, $D$);
 2. Visit $SG$ in depth first order and for each element $e$,
      calculate the total coverage it has for all the
      elements within its subtree, store as $e.cov$;
 3. Initialize $CS = SG.\mathcal{E}$, $DS = \emptyset$;
 4. for each pair of elements $e_1, e_2 \in SG.\mathcal{E}$:
 5.   if $e_1, e_2$ are ancestor-descendants (let $e_1$ be the ancestor):
 6.     select $e_c$ from among the elements $e_1$ directly
          connects as the one with highest coverage of $e_1$;
 7.     $C_1 = e_2.cov$; // see Theorem 1 for meanings of $C_1, C_2$
 8.     $C_2 = C_{e_1 \to e_2} * C_1$;
 9.     foreach element $e'$ along the path from $e_1$ to $e_2$:
10.       if $C_{e_2 \to e'} > C_{e_1 \to e'}$: update $C_1, C_2$ based on $e'.cov$;
11.     determine if $e_1$ dominates $e_2$ based on Theorem 1;
12.     if $e_2$ is dominated: remove $e_2$ from $CS$; add $(e_1, e_2)$ to $DS$;
13. foreach set $E$ of $K$ elements from $CS$:
14.   compute its summary coverage
Output: $E$, the set of $K$ elements with highest summary coverage
          among mutually non-dominant elements;
        $DS$, the set of element pairs with dominance relationship.
```

Figure 6: Algorithm MaxCoverage.

each `profile` node will have multiple `interest` nodes associated with it, for example). We can examine whether element `@category` is dominated by element `interest`, where E={`@category`}, e_c=`profile`. Assume both conditions are satisfied. For a summary containing `@category`, but not `interest`, the summary coverage contributed by `@category` is the coverage of itself, plus the coverages of all elements represented by `@category` in the summary. There are two scenarios: `@category` represents `interest` or it does not, which depends on whether `profile` is in the summary. In the first scenario, both `@category` and `interest` are now represented by `interest` in the new summary, and the decrease in the coverage of `@category` will be offset by the increase in the coverage of `interest` according to the first condition. Furthermore, if there is any other element represented by `@category` (e.g., `education`) in the old summary, the new summary will have its coverage increased further. In the second scenario, in the worst case, `interest` is represented by `profile` in the summary. Due to the second condition, the increase in the coverage of `interest` by switching from `profile` to `interest` again offsets the decrease in the coverage of `@category`. Therefore, we can be assured that the replacing `@category` with `interest` in the summary will produce a higher coverage.

The dominance relationship reflects significant overlap between the coverages of two schema elements, and can be used to greatly reduce the number of candidate schema elements to be examined. However, pairwise computation of dominance between each pair of schema elements is in itself still expensive. We observe that a dominance relationship is most often due to an element being dominated by an ancestor[6]. Therefore, we restrict the search for dominance between elements with ancestor-descendant relationships. As a result of this heuristic, we may miss some dominance relationships, and unnecessarily retain some dominated elements, but this will not decrease the quality of the final result computed. Figure 6 sketches the algorithm for obtaining the set of K elements that has the highest combined coverage.

[6]Ancestor in terms of both structural links and value links. For values links, the *referee* element is considered equivalent to a *parent* element.

```
Algorithm BalanceSummary:
Input: Schema $SG$, Database $D$, summary size $K$
 1. $SG$ = annotateSchema($SG$, $D$);
 2. Initialize $E = \emptyset$;
 3. Obtain list $I$ of schema elements ordered by importance
      using Algorithm MaxImportance;
 4. Obtain set $DS$ of element pairs with dominance relationship
      using Algorithm MaxCoverage;
 5. foreach element $e \in I$ in the order of importance:
 6.   if $e$ is dominated by some $e' \in E$ according to $DS$:
 7.     continue; // $e$ is skipped
 9.   else if $e$ dominates $e' \in E$ according to $DS$:
 0.     remove $e'$ from $E$;
 0.     add all elements skipped due to $e'$ back to $I$;
 8.   else add $e$ to $E$;
 9.   if size of $E = K$: break;
Output: $E$, the set of $K$ elements for the balanced summary.
```

Figure 7: Algorithm BalanceSummary.

Complexity: The time complexity for the pruning component is $O(dpN)$, where d is the average depth of a schema element and p is the average number of paths a schema element is involved. The time complexity for examining the remaining candidate sets is $O(KN^2\mathcal{C}^K_{N'})$, where N' is the number of schema elements not dominated by any other element. We typically see over 50% reduction from N to N'.

4.4 Balancing Importance and Coverage

Maximizing both importance and coverage is often impossible and there is no single optimal way to balance the two desired criteria. We achieve this balance by adopting a heuristic, which selects elements that are important while avoiding having two elements in the summary where one element dominates the other in coverage. The algorithm is shown in Figure 7 and is straightforward. The time complexity of the algorithm, according to previous analyses, is $O(N\log N + (c+dp)N)$ – note that we only need to generate the dominance pairs in Algorithm *MaxCoverage*.

5. EVALUATING SCHEMA SUMMARY

While the usefulness of summarization for complex schemas is intuitive, measuring the quality of a schema summary is a non-trivial task. One of the ultimate measurements of the summary quality is how much benefit it can provide to the user in her querying for the desired information. This can be indirectly measured by asking expert users, who are knowledgable about the schema and the frequently asked queries, to manually generate "good" summaries (i.e, summaries that they believe can best help them in querying). This creates a "gold standard" set of summaries, against which our automatic summaries can be compared and analyzed (Section 5.2). Still, an objective measurement of quality is desired because it can provide an automatic way of evaluating schema summaries. This allows us to assess the quality of a large number of schema summaries of varying properties and can help engineer a better summarizer. Toward this goal, we design a schema summary quality metric, *query discovery cost*, based on the benefit a schema summary can provide to the user in formulating queries that are consistent with the database schema (Section 5.3). We then perform a comprehensive analysis of our summarization system and show that summaries generated by the system can significantly reduce the *query discovery cost* (Section 5.4). The details of the datasets and summaries generated here are also available at the project website[7].

[7]http://www.eecs.umich.edu/db/schemasummary/

	XMark	TPC-H	MiMI
# Schema elements	327	70	155
# Data elements (in 000s)	1,573	12,550	7,055
# Queries	20	22	52
Avg. query intention size	3.65	13.4	3.35

Table 1: Dataset statistics.

5.1 Datasets

We start by introducing the datasets being used for evaluation. In fact, finding appropriate datasets is itself a nontrivial task. A suitable dataset must: 1) come with both data and schema information; 2) be complex enough for its summaries to be meaningful, and 3) have a set of known queries associated with it. While there are many publicly available sets of real data, it is difficult to obtain associated query sets for these. Benchmark datasets come with query sets, but are usually designed with simple schema since the emphasis is on data manipulation performance. In the end, we chose two benchmarks, one XML and one relational, and one real dataset, with relatively complex schema, for which we had the query trace available. Specifically, they are: 1) **XMark**, an XML benchmark dataset [12] derived from an auction site, with a set of 20 queries. We have also used this as a running example in this paper. 2) **TPC-H**, a relational benchmark dataset [14] for a decision support system, with a set of 22 queries. 3) **MiMI**, a real world scientific dataset [10] on protein interaction information that is built and deployed by us at University of Michigan. MiMI integrates a variety of biological data sources (with emphasis on protein information) and provides a common schema to users for uniform access to the underlying heterogeneous data. MiMI has been deployed online since July 2005 and we collected query trace from August 2005 to January 2006. A total of 2167 queries were issued by users during this 6-month period, among which 1024 queries were distinct. We further clustered the queries by consolidating two queries into the same cluster if they differed only in their variable names or constants in the conditions. This resulted in a total of 52 query groups and we selected one query from each group for evaluation. The detailed statistics about the datasets are shown in Table 1[8].

5.2 Comparison with Expert Summaries

We enlisted two sets of expert users (three each) to generate "gold standard" summaries for the MiMI and XMark datasets. For MiMI, the experts are simply the administrators. For XMark, the experts have worked with the XMark benchmark extensively. Summaries of different sizes were requested: 5, 10, 15 elements. Correspondingly, we generated automatic summaries at the same sizes and measured the agreement between those and the expert summaries. The agreement between two schema summaries is defined as the percentage of the number of elements selected by both the user and the system over the summary size. Finally, a user *consensus* summary of a particular size is generated by combining all user summaries and retaining only elements selected by a majority of the users (in this case, at least two users). This experiment was not done for the TPC-H dataset because users familiar with it were not locally available.

[8]For synthetic datasets, we intentionally selected medium scale factors for data generation: 1 and 0.1 for XMark and TPC-H, respectively. The reason is that for those datasets, the scale factor does not affect the relative distribution of the data and thus it has no impact on the summarization.

XMark	5-element	10-element	15-element
User 1 vs. Auto.	100%	70%	67%
User 2 vs. Auto.	60%	80%	67%
User 3 vs. Auto.	100%	80%	87%
User Agreement	60%	50%	53%
Consen. vs. Auto.	100%	70%	80%
MiMI	5-element	10-element	15-element
User 1 vs. Auto.	100%	90%	87%
User 2 vs. Auto.	80%	70%	67%
User 3 vs. Auto.	80%	90%	87%
User Agreement	80%	70%	60%
Consen. vs. Auto.	80%	80%	87%

Table 2: Agreement between automatic summaries and expert summaries on XMark and MiMI datasets. User agreement measures the percentage of elements all three experts agree on.

Results: The results are shown in Table 2. We see that humans do not always agree on what is the best summary, though they do tend to have quite a bit of commonality. Furthermore, the percentage in common decreases slightly as the size of summary increases – humans agree more on what are the most important aspects of the database, but somewhat less on the next level. Our system was in reasonable consonance with human experts: the difference between the system and any human was no greater than that between pairs of humans. In short, our automated "summarizer" was able to produce summaries at different sizes that appear to be similar to what a human may have produced.

5.3 Query Discovery Cost

Comparing automatic summaries with expert summaries is always subjective and the results may often be considered anecdotal. In this section, we present a metric for objective evaluation of the quality of schema summaries. The metric is based on the notion of *Query Discovery*, which models how an ordinary user, defined as one with little schema knowledge about the database, formulates a concrete query specification given the information need she has in mind. We assume each ordinary user to have an implicit *query intention* as her information need, and she is then required to explore the schema (or schema summary) to convert this implicit intention into explicit specifications conforming to the schema. In our model, query intention is represented as a set of schema elements the user would like to query upon but doesn't know where in the schema they are exactly. Query discovery is then the process of finding the exact locations (i.e., paths) for those schema elements. For example, given the following XMark query expressed in English:

Return the name of the person with id 'person0'.

The query intention is {`person,name,id`}, and the query discovery process is to find the schema paths for these elements so that a complete XQuery specification can be generated as the following:

```
for $b in doc("auc.xml")/site/people/
            person[@id='person0']
return  $b/name/text()
```

The effort required on the user to locate these schema elements is a reflection of the availability and quality of query discovery support available. Specifically, we assume that the user "visits" one schema element at a time, and charge one unit for each element visited that is not in the query (intention). The cost of query discovery for a particular query is the total charge accumulated by the time all elements in the query intention are visited.

A naive approach to schema exploring is to scan through all the elements until the ones of interest are found. However, if there is some structure to the schema, one can do better. XML schema, for example, has a hierarchical structure and a tree-traversal from the root following structural links is more appropriate. We implemented depth-first pre-order and breadth-first pre-order strategies as a baseline. For relational schemas, which do not contain structural links (except those from the system-introduced root), we can adopt value links for traversal instead. Furthermore, the label of an element at an intermediate node in the schema graph will be indicative of the nature of elements to be found in the sub-tree rooted at it. E.g. someone looking for the end date of an auction is not likely to look under `person`. A good tree traversal need to make use of such information and explore first the sub-tree that looks most promising, in a "best-first" strategy. To obtain a firm cost for such a traversal we make the optimistic assumption that the label of each sub-tree root perfectly indicates whether an element of interest to the user is in the sub-tree. Thus, in a pre-order best-first traversal, the user starts from the root, and at each stage examines children of the current node one at a time until it finds one that it should visit. This child node now becomes the current node and the process above repeats.

Query discovery with schema summary: A schema summary provides support for query discovery by presenting early on, to the user, elements that are more likely to be queried, and elements that are more closely related to other schema elements likely to be queried. As a result, users can locate schema elements of interest easier. Query discovery with a schema summary proceeds just as with regular schema, except that now the traversal also includes abstract elements in addition to original elements. When an abstract element of interest is visited, it can be expanded, and the enclosed original elements visited. One unit of cost is applied to every abstract element visited as well as to every original element visited that is not in the query intention.

Limitations: There are several limitations to the proposed query discovery cost metric that cause our cost estimate to deviate from the true user effort involved in the process of query formulation. First, our model assumes that the user knows the labels of the schema elements in their query intention[9]: it's the schema locations of those elements that are unknown. In practice, a naive user may not know the exact labels of those elements and will need to compare multiple schema elements before the right one can be selected. Second, it assumes that the user can unerringly determine whether a schema element of interest is a descendant of a candidate (abstract) element being examined. In practice, users will make some wrong decisions and pursue paths that do not lead to any element of interest. Both limitations are caused by user behaviors that are difficult to quantify with an objective metric. While we do not attempt to consider these in our model, we note here that both limitations lead our best first traversal model to under-estimate the true

[9]Actually, the model assumes that a user will know that a schema element is a desired one as soon as she visits it, which has the same effect.

Avg. cost	XMark	TPC-H	MiMI
w/o summary	-	-	-
Depth First	75.35	74.95	50.27
Breadth First	37.15	67.36	30.23
Best First	11.90	18.41	10.38
w/ summary	6.65	12.05	3.90
size (Summ.%)	10 (3.1%)	5 (7.0%)	10 (6.5%)
# Rounds	160	45	426
Saving%	44.1%	34.5%	62.4%

Table 3: Average cost of Query Discovery w/o and with schema summaries. Saving% are compared with cost w/o schema summary using best first strategy.

cost of query discovery, and we believe their impact is at least comparable for query discovery without schema summary and with schema summary, if not more on the former; because a user is more likely to visit unnecessary schema elements in the original schema than in the schema summary, which is much smaller than the former. Finally, our model does not consider the effort required after the desired schema elements are located, i.e., fully formulating the query according to the target query language and the desired logic. This cost is universal to all query discovery processes, with or without schema summary, and is therefore outside the scope of this study.

5.4 Summary Benefits for Query Discovery

For each dataset and each query in its query set, we manually generated the query intention comprising the set of schema elements referenced in the query. This set can be extracted from the English description of the query (e.g., XMark) or be reverse engineered from the actual query itself (e.g., TPC-H and MiMI). We then computed the number of original/summary elements that must be explored to locate the schema elements in the query intention.

Results: Table 3 shows the benefits of automatic summaries that are generated at the sizes of 5 (for TPC-H) and 10 (for XMark and MiMI) using Algorithm BalanceSummary. We see that depth-first is a poor strategy for query discovery – much cost can be incurred traversing irrelevant portions of schema space. Breadth-first is not too bad, but best-first is substantially better. Summary decreases the cost of query discovery (using best-first) by about a factor of two. We observe that the query discovery cost is substantially higher for the TPC-H dataset than for the other datasets. This is because the queries on TPC-H involve a substantially higher percentage of schema elements (Table 1). Because of this, the opportunity for cost reduction is less, making schema summary not quite as effective as for the other cases. Schema summarization was most effective for the one real data set we used, compared to the two benchmarks. We believe this is because benchmarks, by design, "spread their queries" around the schema, whereas real queries on real databases tend to focus on the important elements. However, our experiments do not provide enough information to verify this conjecture. In the next four sections, we provide a comprehensive evaluation of the automatically generated summaries by analyzing the impacts of various parameters affecting the summary generation.

Impact of Summary Size: Different schema summary sizes lead to different levels of effectiveness in supporting query discovery, which are reflected in the query discovery

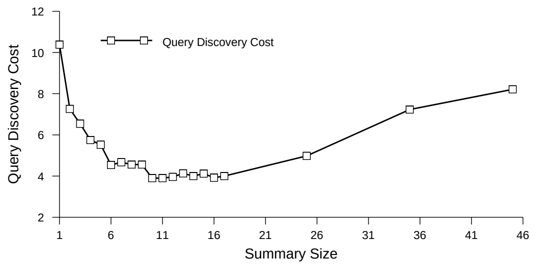

Figure 8: Impact of summary size on Query Discovery.

costs. We examine such impact by evaluating summaries of various sizes using the MiMI dataset. The result is shown in Figure 8. As expected, a summary with very small size (less than 5 elements, or 3% of schema size) does not achieve full effectiveness in decreasing query discovery cost – substantial amount of information has been abstracted away. As the size grows, the summary approaches its full effectiveness (lowest query discovery cost) and stays relatively stable in this effectiveness for a fairly large range (9-17 elements). As the size grows further, too many elements are being included in the summary, which in fact start reducing the effectiveness. The cost continues to increase along with the summary size, until it reaches the full schema cost.

Schema Structure and Data Distribution: We now consider the effect of the way importance is calculated. By varying the neighborhood factor p and other parameters in Formula 1, the algorithm MaxImportance (which provides BalanceSummary with the list of important elements) can be adjusted between being fully data driven (i.e., ignoring schema structure) to fully schema driven (i.e., ignoring data distribution). *Full Data Driven*: $p = 1$. By setting p to 1, all of the importance is determined by the data distribution. Elements are selected into the summary according to their cardinalities in the database, regardless of the schema structure. *Full Schema Driven*: $\forall i, j, RC(i \rightarrow j) = 1, \forall i, Imp^0_{e_i} = 1$. By setting relative cardinalities between any two schema elements to 1 and ignoring the initial data distribution, the model will now consider only the schema structure. Elements are included into the summary according to their connectivity in the schema, regardless of the data distribution. We also evaluated the quality of the summary for a wide range of p values between the two extremes. We found, for all three datasets, that the relative importance of elements remained the same for all values of p between 0.1 and 0.9. As a result, the schema summary generated remained stable as well. We show the results for a representative mid-point with $p = 0.5$ (data-and-schema driven) and the above two extreme strategies in Figure 9. We find that data driven summarization works very poorly for XMark, and schema driven summarization works very poorly for MiMI. However, the complementary summarization is effective in both datasets. Overall, the data-and-schema driven summary is much more effective than a summary based on either data or schema alone.

We also note here that when p is close to 0, the system takes a long time to converge on the final importance value, providing one more reason not to choose too small a p value. Our suggestion is to use a p value around 0.5 in practice – the results should not be too sensitive to the precise value chosen for this parameter.

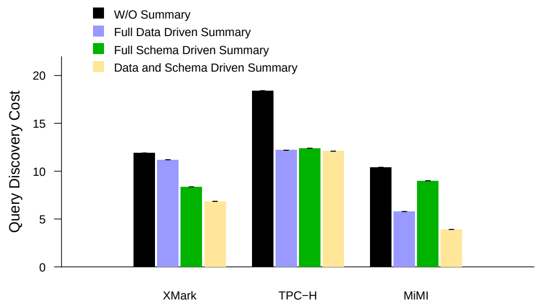

Figure 9: Impact of schema structure and data distribution on Query Discovery cost. Summaries are of size 5 for TPC-H and 10 for XMark and MiMI.

Avg. cost	XMark	TPC-H	MiMI
w/o summary	11.90	18.41	10.38
Summ. size (summ.%)	10 (3.1%)	5 (7.0%)	10 (6.5%)
w/ BalanceSummary	6.65	12.05	3.90
Saving%	44.1%	34.5%	62.4%
w/ MaxImportance	8.35	12.36	5.56
Saving%	29.8%	32.9%	46.4%
Saving Reduction%	32.4%	4.6%	25.6%
w/ MaxCoverage	10.20	12.18	5.78
Saving%	14.3%	33.8%	44.3%
Saving Reduction%	67.6%	2.0%	29.0%

Table 4: Impact of balancing importance and coverage on Query Discovery cost. Summaries are of size 5 for TPC-H and 10 for XMark and MiMI.

Balancing Importance and Coverage: We further evaluated whether balancing importance and coverage is indeed necessary. We generated summaries using one of the three algorithms: MaxImportance, MaxCoverage, and BalanceSummary, and examined their query discovery costs. The results are shown in Table 4. As expected, when only one of the two factors are considered, the effectiveness of the summary is significantly reduced for XMark and MiMI datasets: the reduction percentage ranges from 25% to 70%, validating that the two factors are indeed complementary and both are necessary to generate a good summary. The reduction for TPC-H dataset is trivial, this is due to the fact that, for this particular dataset, the set of elements with highest importance values significantly overlaps with the set of elements with highest coverage, leading to almost the same summary being generated by all three algorithms.

Data Evolution: We also examined the impact of data evolution on the resulting schema summary. The MiMI dataset has evolved over time to incorporate more and more data sources, these archived versions of MiMI dataset allow us to compare summaries generated based on different data distributions and check the stability of the summaries. As we can see from Table 5, the summaries being generated remain stable with the evolving database. During October 2005, information regarding protein domains were imported into the database, and the summaries evolved accordingly, and as desired, to reflect this change.

Comparison with ER Model Abstraction: While our work is the first to study relational or XML schema summarization, conceptual schema (ER diagram) abstraction (clus-

	change%	5-ele.	10-ele.	15-ele.
Apr 04 vs. Jan 05	50%	100%	100%	100%
Apr 04 vs. Now	75%	100%	90%	87%
Jan 05 vs. Now	17%	100%	90%	87%

Table 5: Agreement between summaries on different versions of MiMI dataset. Current version is Jan 2006.

Avg. cost	MiMI
with BalanceSummary (Saving%)	3.90 (62.4%)
TWBK [13] w/o human (Saving%)	*9.32 (10.2%)*
TWBK [13] with human (Saving%)	4.38 (57.8%)
CAFP [4] w/o human (Saving%)	*8.56 (17.5%)*
CAFP [4] with human (Saving%)	3.90 (62.4%)

Table 6: Comparing against ER model abstraction techniques on MiMI. Summaries are of size 10.

tering) has a long history. We examined the applicability of its representative techniques, namely [13] and [4], in our schema summarization scenario. One challenge immediately facing us is to provide, for both techniques, semantic labels to the structural and value links, such that the links can be assigned proper weights, which are used to estimate element closeness. While some labeling can be done unsupervised by exploring linguistic techniques and data information, most can not be done automatically. In fact, even humans can find the labeling task difficult and ambiguous. Nevertheless, once the labeling is done properly (with significant human efforts), those techniques do produce schema summaries of comparable qualities. Table 6 shows the results of comparing those techniques (with or without human efforts) against our system in summarizing the MiMI schema.

Discussion: First, in addition to schema structure and data distribution, another potentially important input to automatic schema summarization algorithms is historical queries. By analyzing the query history, important elements can be extracted as the most frequently queried elements. While we believe query history will be useful, we also note that it is typically not available when a database is newly created, and it is often slow to adapt to the changes in the schema or database. Second, due to space limitation, we do not provide detailed performance information here. We will simply report that the schema annotation process runs in time linear to the database size, while the actual summarization process finishes within 5 minutes on a 2.0GHz P4 PC with 1GB RAM for all three datasets evaluated.

6. RELATED WORK

In [5], Feldman and Miller proposed *entity model clustering*, a systematic methodology for manually grouping entities into clusters according to semantic closeness. The technique was extended and improved upon by Teorey et al in [13], where various grouping operations were further defined. The amount of human effort required in those techniques, as discussed before, is significant, making them unsuitable for a heterogeneous and evolving environment. Automated systems for entity clustering have since been proposed, including [7, 6, 1, 3, 4, 11]. Most of those systems rely heavily on the semantics embedded in the relationships to guide the process and are therefore not truly automated. There are few evaluation methodologies of ER abstractions, with one notable exception in [11], where the authors proposed nine principles for judging the goodness of an ER diagram decomposition. Most of the principles, however, are features naturally resulting from good clustering algorithms and may not correspond to the actual benefits to the user. Finally, the iterative formula for importance calculation are inspired by Google PageRank [2] and the Similarity Flooding algorithm by Melnik et al [9].

7. CONCLUSION

Modern database systems can often have complex schema. In this paper, we have proposed a notion of schema summary to manage this complexity. We have suggested importance and coverage as two relevant properties by which to judge the quality of a schema summary. We have presented an algorithm to automatically compute high quality schema summaries, using notions of summary importance and coverage. An experimental assessment of our summaries, both subjectively, and objectively using a query discovery cost metric we define, shows that our algorithm is able to find good summaries for a given schema and database. An outline or overview is key to human understanding of complex material. The notion of schema summary we suggest in this paper has the potential to be a really valuable means to introduce humans to new databases. We believe that schema summaries can also be helpful in schema matching, and intend to explore this issue in the future.

8. REFERENCES

[1] J. Akoka and I. Comyn-Wattiau. Entity-Relationship and Object-Oriented Model Automatic Clustering. *Data Knowl. Eng.*, 20(2):87–117, 1996.

[2] S. Brin and L. Page. The Anatomy of a Large-Scale Hypertextual Web Search Engine. *Computer Networks and ISDN Systems*, 30(1–7):107–117, 1998.

[3] L. J. Campbell, T. A. Halpin, and H. A. Proper. Conceptual Schemas with Abstractions: Making Flat Conceptual Schemas More Comprehensible. *Data Knowledge Engineering*, 20(1):39–85, 1996.

[4] S. Castano, V. D. Antonellis, M. Fugini, and B. Pernici. Conceptual Schema Analysis: Techniques and Applications. *TODS*, 23(3):286–333, 1998.

[5] P. Feldman and D. Miller. Entity Model Clustering: Structuring a Data Model by Abstraction. *The Computer Journal*, 29(4), 1986.

[6] C. Francalanci and B. Pernici. Abstraction Levels for Entity-Relationship Schemas. In *ER*, 1994.

[7] S. B. Huffman and R. V. Zoeller. A Rule-Based System Tool for Automated ER Model Clustering. In *ER*, 1989.

[8] S. Melnik. *Generic Model Management: Concepts and Algorithms.* 2004. Chapter 7, Ph.D. Dissertation, University of Leipzig.

[9] S. Melnik, H. Garcia-Molina, and E. Rahm. Similarity Flooding: A Versatile Graph Matching Algorithm and Its Application to Schema Matching. In *ICDE*, 2002.

[10] MiMI. http://mimi.ctaalliance.org.

[11] D. Moody and A. Filtman. A Methodology for Clustering Entity Relationship Models – A Human Information Processing Approach. In *ER*, 1999.

[12] A. R. Schmidt, F. Waas, M. L. Kersten, D. Florescu, I. Manolescu, M. J. Carey, and R. Busse. The XML Benchmark Project. Technical Report INS-R0103, CWI, April 2001.

[13] T. Teorey, G. Wei, D. Bolton, and J. Koenig. ER Model Clustering as an Aid for User Communication and Documentation in Database Design. *Comm. of ACM*, 32(8), 1989.

[14] TPC-H Benchmark. http://www.tpc.org/tpch/.

[15] T. S. Tullis. *Handbook of Human-Computer Interaction.* Elsevier, Amsterdam, Netherland, 1988.

[16] W3C. XML Schema. http://www.w3.org/TR/xmlschema-0/.