Comprehension-Based Result Snippets

Abhijith Kashyap
University of California at Riverside
Riverside, CA, USA
akash001@ucr.edu

Vagelis Hristidis
University of California at Riverside
Riverside, CA, USA
vagelis@cs.ucr.edu

ABSTRACT

Result snippets are used by most search interfaces to preview query results. Snippets help users quickly decide the relevance of the results, thereby reducing the overall search time and effort. Most work on snippets have focused on text snippets for Web pages in Web search. However, little work has studied the problem of snippets for structured data, e.g., product catalogs. Furthermore, *all* works have focused on the important goal of creating *informative* snippets, but have ignored the amount of user effort required to *comprehend*, i.e., read and digest, the displayed snippets. In particular, they implicitly assume that the comprehension effort or *cost* only depends on the length of the snippet, which we show is incorrect for structured data.

We propose novel techniques to construct snippets of structured heterogeneous results, which not only select the most informative attributes for each result, but also minimize the expected user effort (time) to comprehend these snippets. We create a comprehension model to quantify the effort incurred by users in comprehending a list of result snippets. Our model is supported by an extensive user-study. A key observation is that the user effort for comprehending an attribute across multiple snippets only depends on the number of unique positions (e.g., indentations) where this attribute is displayed and not on the number of occurrences. We analyze the complexity of the snippet construction problem and show that the problem is NP-hard, even when we only consider the comprehension cost. We present efficient approximate algorithms, and experimentally demonstrate their effectiveness and efficiency.

Categories and Subject Descriptors

H.5.2 [**Information Interfaces and Presentation**]: User Interfaces – *user-centered design, graphical user interfaces.*

Keywords

Query Interfaces, Information Overload, Result Snippets.

1. INTRODUCTION

A large number of databases are heterogeneous in nature. Examples of such databases include product catalogs (Amazon, eBay etc.) and medical data (Stanford diabetes study, patient records, Human Genome project), among many others. Such heterogeneous data is characterized by a high structural variance amongst the objects in the database, where objects have different and usually overlapping sets of attributes. As an example, consider the Amazon product catalog which consists of objects of different types including *Laptop, Desktop* and *Camera etc*. Each object type is associated with different and possibly overlapping schemas. For instance, *Laptop* has attributes *Price, Display Size*, *Fingerprint Reader* etc., whereas *Camera* has attributes such as *Price, Shutter Speed, Zoom* etc. As another example, a patient record stores various types of data, e.g., a diabetic patient's record includes *Blood Pressure*, *Blood Sugar* level and *Insulin Dosage,* whereas a patient's record with *Osteoporosis* includes *Bone Density*, *Calcium Level,* etc.

Keyword search is the dominant query interface in most such systems. Hence, naturally the query answer typically consists of a list of heterogeneous objects, due to the ambiguous nature of keyword queries. Query interfaces use a variety of methods to help users find the results that they are most interested in, like ranking [1-3], categorization (facets) [4-6], and result snippets.

A snippet is a summary of the information contained in a result and its purpose is to help the user make a decision about the relevance of the given result. As an example, consider snippets for the results of the query *'acer laptop 3gb'* on an e-commerce Web site, as shown in Figure 1. Amazon.com, as most other similar systems, uses a fixed hardcoded snippet schema for each type of object, as in Figure 1b. In particular, Amazon.com always displays attributes *Brand, Model, Display, Model Name* and *Color* for *Laptop* results (only *Brand* and *Display* are in our result-set). This is clearly suboptimal, since such snippets do not always allow differentiating among the displayed results ([7] makes this point for XML results). For example, the snippets of two *Laptop* results (#1 and #3) in Figure 1b show the same attributes and values. Instead, it is beneficial to include a discriminating attribute for Acer laptops (e.g. *Cover Material*) in the snippet, which may not be important for other results in the query. Further, this fixed-schema approach is inefficient for diverse result-sets [8-10] in which the results have few attributes in common.

The only work we are aware of on snippets construction for structured data, studies snippets of XML results, has focused on the *informativeness* of the snippets, which describes how useful the information on the snippets is to help the user select a result, e.g., how representative or distinguishable the snippets are [7]. For instance, if many results have *Display Size=11.3"*, this information should be displayed on the snippets. Figure 1c shows a list of snippets generated with informativeness in mind[1]. Comparing Figures 1b and 1c, we observe that the snippets in Figure 1b appear to have a somewhat uniform schema (at least for items of the same type), while the ones in Figure 1c look very jagged and disorganized. The tradeoff is that the more uniform snippets are easier to read, while the disorganized ones may offer more useful information about the returned results. The goal of snippets is to minimize the user effort (time) in finding the results of interest. Hence, we argue that *the user time spent reading the snippets is important as is the information on the snippets*.

[1] For this example, you can think of informativeness as the amount of information. We provide more details in Section 4.

ID	Type	Brand	Memory	Capacity	Processor	Display	Cover Material	Price
1	Laptop	Acer	4Gb		Intel i5	13.3”	Aluminum	$1299
2	Tablet	Acer		16Gb	Nvidea Ion	7”	Gorilla Glass	$380
3	Laptop	Acer	3Gb		Intel i5	13.3”	Carbon Fiber	$1250
4	HDD	Seagate		500Gb			Plastic	$200
5	Memory	Kingston	2Gb					$100
6	Memory	Corsair	4Gb					$150

(a) Result-set of query ‘*acer laptop 3gb*’

1	*Brand*: Acer		*Display:*13.3”
2	*Brand*: Acer	*Capacity*: 16Gb	*Display*: 7”
3	*Brand*: Acer		*Display:*13.3”
4	*Brand*: Seagate	*Capacity*: 500Gb	
5	*Brand*: Kingston		
6	*Brand*: Corsair		

(b) Fix Schema Snippet, with high information loss

1	*Memory:*4Gb	*Processor:* Intel i5	*Price:* $1299
2	*Capacity*:16Gb	*Display:* 7”	*Cover:* Gorilla Glass
3	*Brand*: Acer	*Processor:* Intel i5	*Display:* 13.3”
4	*Capacity:* 500Gb	*Cover*: Plastic	*Price*: $200
5	*Brand*: Kingston	*Memory*: 2Gb	*Price*: $100
6	*Brand*: Corsair	*Memory*: 4Gb	*Price*: $150

(c) A very informative, but hard to read snippet

1	*Cover:* Aluminum	*Memory:*4Gb	*Processor:* Intel i5
2	*Brand*: Acer	*Display:* 7”	*Processor:* Nvidea Ion
3	*Cover:* Carbon Fiber	*Display:* 13.3”	
4	*Brand*: Seagate	*Capacity:* 500Gb	*Price*: $200
5	*Brand*: Kingston	*Memory:*2Gb	*Price*: $100
6	*Brand*: Corsair	*Memory:*4Gb	*Price*: $150

(d) An informative and easily comprehensible snippet

Figure 1. A heterogeneous result-set for query ‘*acer laptop 3gb’* and three snippets with different characteristics

Figure 1d shows an example of a snippet that is both highly *informative* (contains a number of attributes) and is easy to *comprehend* since the attributes in snippets of results are aligned and are therefore easy to read.

No work has studied the *comprehension cost* of structured snippets, which is the user effort required to read and digest the information displayed by the snippets. We argue that the comprehension cost should be taken into consideration during the snippets generation process. In this paper, we propose a methodology to construct snippets that simultaneously minimize the comprehension effort and information loss (i.e. maximize informativeness). Our work achieves this goal as follows:

First, we propose the first model for the user comprehension cost of reading structured snippets. We perform user surveys that confirm our intuition that by indenting the snippet attributes in a way that common attributes have the same position across the snippets we reduce the user comprehension cost. In particular, we show the surprising result that *the comprehension cost for an attribute does not depend on the number of snippets that contain it, but only on the number of different positions where it appears in the snippets*.

Next, we define a quantitative model for the information content (termed *informativeness)* of snippets with respect to complete results. We leverage previous work on snippet informativeness and adapt it to structured objects.

We analyze the problem of constructing optimal snippets, i.e., minimizing both the comprehension cost and the information loss, for a list of results and show that this problem is NP-hard. We present efficient algorithms for snippet construction and evaluate their performance and efficiency.

Contributions: We make the following specific contributions:

1. A cost model that quantifies the comprehension effort of a user reading a list of result snippets. We conducted user surveys using Amazon Mechanical Turk to validate the model and estimate its parameters (Section 3).
2. Naturally, a snippet only contains a subset of the result’s attributes. We build upon previous works that characterize the information content or informativeness of a snippet and present several measures for structured data (Section 4).
3. A proof that the problem of constructing *optimal* snippets, i.e. snippets that simultaneously minimize comprehension cost and maximize informativeness is NP-Hard (Section 5).
4. Heuristic algorithms to efficiently construct highly informative snippets which have low comprehension cost (Section 6).
5. An experimental evaluation to validate the effectiveness of the constructed snippets using real datasets (Section 7).

Section 2 defines the problem. Related work is discussed in Section 8 and we conclude in Section 9.

2. FRAMEWORK AND DEFINITIONS

In this section, we formally define the snippets construction problem. We start by defining the data model.

Database: The database is a single relation D with m attributes $A = \{A_1, \dots, A_m\}$. Each attribute A_i has an associated active domain $ADom(A_i)$ of un-interpreted constants, which includes the *null* value. The database D is sparse and heterogeneous, i.e. tuples $r \in D$ have values for different subsets of A, and have the null value for the rest of the attributes. We use $A^r \subseteq A$ (typically $|A^r| \ll |A|$) to denote the set of attributes of a tuple r .

Result-set: A user exploring D typically submits a query and the system returns a ranked result-set $R = \{r_1, \dots, r_n\} \subseteq D$ of objects (we use the terms object, tuple and result interchangeably depending on the context).

Example: Figure 1a shows a subset of the results of query *'acer laptop 3gb'* on Amazon.com. Only seven (we do not count *Type* as an attribute) of nearly hundred attributes are shown. The query returns not only *Laptops* (indicated by the *Type* attribute), but also results of type *Memory*, *Tablets* and *Hard Disk Drives*. The schema of each object depends on its *type* and can have common attributes with other types (e.g. *Price*), but also different attributes (e.g *Laptops* and *Tablets* have a value for attributes *Display* and *CPU*, whereas *Memory* does not).

A snippet with many attributes can be difficult to present and can overwhelm the user with details. Therefore we typically require that the size of each snippet be bounded to k attributes. The snippets in Figures 1b-d have size $k \leq 3$. Note that the size of a snippet could also be defined in other ways like the number of characters. However, we have found that the number of attributes offers a reasonable bound that also allows a structurally uniform presentation (e.g., in tabular form).

Result Snippet: A snippet $s(r)$ for a result r is a k-tuple $< A_1 = a_1, \dots, A_k = a_k >$ [2], where $A_i \in A^r$ is an attribute of r or is empty, and $a_i \epsilon Dom(A_i)$ is a value. To simplify the presentation, we often denote $s(r)$ as $< A_1, \dots, A_k >$ and use $|s(r)|$ to denote the number of attributes in $s(r)$. E.g., the first snippet r in Figure 1b is <*Brand, null, Display*>, and $|s(r)| = 2$.

Result-set Snippet: A result-set snippet $S(R,k)$ of a result-set $R = \{r_1, \dots, r_n\}$ is $S(R,k) = \{s(r_1), \dots, s(r_n)\}$ where $s(r_i)$ is the snippet of result r_i, and $|s(r_i)| \leq k$.

Figures 1b-d show example result-set snippets of the result-set in Figure 1a. It is possible that results of same type have different attributes in their snippets, e.g. the first and the third *Laptop* snippets in Figure 1c. To construct $S(R,k)$, a subset $A^s \subseteq A^r$ of size at-most k has to be selected for each result r, and the attributes in A^s have to be ordered as a k-tuple. The order (position) of the attributes is an important factor for the comprehension cost, as we explain in Section 3.

Result-set Snippet Construction Problem: Before formally defining the problem, we must define what a good result-set snippet is. Let function $\mathcal{F}(S,R,k)$, which will be defined below, be the goodness of $S(R,k)$, given R and k. By goodness, we mean that $S(R,k)$ simultaneously minimizes the *comprehension cost* and maximizes the *informativeness*.

Given a result-set R and snippet size bound k, construct a result-set snippet $S(R,k)$ such that:

$$S(R,k) = argmax_{S'(R,k)}\left(\mathcal{F}(S'(R,k),R,k)\right) \qquad (1)$$

To capture the comprehension effort, we introduce the following function predicate: $Compreh(S,R,k)$ that quantifies the user effort in reading and understanding the result-set snippet $S(R,k)$. Analogously, $Inform(S,R,k)$ captures the informativeness of the result-set snippet.

The goodness of a result-set snippet decreases with increasing comprehension effort, i.e.

$$\mathcal{F}(S,R,k) \propto 1/Compreh(S,R,k)$$

and increases with informativeness, i.e.

$$\mathcal{F}(S,R,k) \propto Inform(S,R,k)$$

To combine the two competing factors, we formulate the snippet construction problem as a bi-criteria optimization problem and introduce a trade-off parameter $\lambda \in [0,1]$. The range (and units) of $Compreh$ is different from that of $Inform$. Therefore, to avoid having the goodness be dominated by a single factor, we choose to define the optimization function as a product, instead of the more common linear combination, as follows:

$$\mathcal{F}(R,S,k) = \frac{Inform(S,R,k)^{\lambda}}{Compreh(S,R,k)^{1-\lambda}} \qquad (2)$$

Intuitively, smaller values of λ lead to result-set snippets with smaller comprehension cost, which translates to fewer unique attributes and stricter alignment of same attributes (Section 3), whereas a large value of λ favors more informative snippets.

3. COMPREHENSION MODEL

In this section we model how users read snippets and present a comprehension cost model. We start, in Section 3.1, by describing the process by which a user reads and understands a list of results and identify the factors that affect comprehension. Next, in Section 3.2, we describe the details of a user study specifically designed to study the effect of the aforementioned factors on comprehension effort. Finally, in Section 3.3, we present the results of this study and use it to formulate a comprehension model that quantifies this effort.

3.1 Comprehension Model and Factors

There have been studies [11-13] in the HCI community on the benefits of tabular presentation of results over a list-based presentation. Users can read tables horizontally −one snippet at a time− or vertically –one column at a time. When users view a result-set snippet, they generally look for attributes of interest and for each such attribute scan all snippets to see its value in other results, in order to get a picture of the result-set. E.g., in Figure 1d, the user may scan the result-set snippet and get interested in attribute *Memory*, and then scan vertically to examine the value of *Memory* in other snippets. Then, the user may pick *Brand* and repeat the process. Eventually, the user will have *comprehended* the result-set snippet, that is, examine all attributes of interest to her, in order to make a decision (e.g. select a result or refine the query). The comprehension cost $Compreh(S,R,k)$ is the total user cost during this process.

A key *hypothesis*, which we test below, is that the effort required to locate the attributes of interest and their values, depends on how they are arranged in the result-set snippet. More specifically, we hypothesize that if all the snippets have the same schema (as in Figure 1b), then the user first determines the position of a given attribute that she is interested in and then reads its values for all snippets. The associated comprehension cost then mainly consists of the cost to locate the attribute position, since reading (comparing) a list of aligned values entails an almost fixed effort, as we show below. However, fixing the schema is not possible for heterogeneous result-sets when *informativeness* (e.g., diversity) must be taken into consideration. This leads to increased user effort and thereby increased comprehension cost.

Another factor that possibly affects comprehension is the number of times an attribute $(say\, A_i)$ appears in the snippets. For example, if an attribute appears multiple times in the snippets, then the user would have to locate each instance of the attribute to satisfy her information need, thereby increasing the comprehension effort. For example, if the user is interested in the *Brand* attribute in Figure 1c, then she would have to locate its

[2] Assuming a prefix of the attributes in A is in s.

three occurrences. Based on the above discussion, we identify two factors that *may* play a role in the comprehension effort:

1. $nOccur(A_i)$: number of times an attribute appears in the result-set snippet, e.g., 3 for *Memory* in Figure 1c.
2. $nPos(A_i)$: number of unique positions of an attribute in the result-set snippet, e.g., 2 (first and second) for *Memory* in Figure 1c.

Hence, the comprehension cost for an attribute A_i in the result-set snippet $S(R,k)$ is a function:

$$Compreh(S,R,k,A_i) = f\big(nPos(A_i), nOccur(A_i)\big) \qquad (3)$$

Note that we overload $Compreh(.)$ from Section 2.

Given the above discussion on vertical scanning of the snippets, the overall comprehension cost for $S(R,k)$ is approximated by the sum of the costs of its attributes, that is:

$$Compreh(S,R,k) = \sum_{occursIn(S,A_i)} Compreh(S,R,k,A_i) \qquad (4)$$

where $occursIn(S,A_i)$ returns true if attribute A_i is in at least one of the snippets in $S(R,k)$.

Equation 4 shows that only two factors affect the comprehension cost. Although this is supported by our user surveys below, we acknowledge that there can be several other factors that affect comprehension cost such as *comprehension difficulty* of attribute names and values. For example, understanding a *Legal Disclaimer* attribute of results requires more effort than understanding the *Color* attribute. Yet another factor could be the format in which attributes are displayed – e.g. highlighting attributes. However, such factors cannot be quantitatively modeled in any straightforward way, e.g. comprehension difficulty is data-dependent, and highlighting depends on the presentation design. We leave the study of such additional factors as future work.

What is left is to study are the properties of function $f(.)$ in Equation 3. As mentioned above, comprehension of snippets is a complex activity involving a number of factors such as locating, reading and understanding the data present in the snippets. The effort or cost of these actions is subjective and is difficult to measure. Instead, we propose to measure the overall effort by measuring the time taken by a user to complete a comprehension task. Next, we describe the user experiment we conducted to measure the effort in comprehending an attribute A_i, namely, $\Big(f\big(nPos(A_i), nOccur(A_i)\big)\Big)$ in a result-set snippet.

Table 1. Combinations of $nOccur(A_i)$ and $nPos(A_i)$. Each cell is a task.

$nOccur(A_i)$	*1*	*4*	*8*	*12*	*15*
$nPos(A_i)$	1	1	1	1	1
$nPos(A_i)$		2	2	2	2
$nPos(A_i)$		4	4	4	4
$nPos(A_i)$			6	6	6

3.2 User Study Setup

To determine the effect of the parameters of our comprehension model, namely number of positions $nPos(A_i)$ and occurrences $nOccur(A_i)$ in snippet $S(R,k)$, and the relationship between them, it is necessary to determine the time it takes for users to comprehend the given attribute for different configurations of these parameters. More concretely, for a snippet size k and a result-set size of n ($> k$), an attribute can be present in snippets of all or some (between 1 and n) of the n results and can be placed in any number of positions between 1 and k. Measuring the time taken by users to *comprehend* the attribute in these multiple configurations gives the estimated *relative* effort.

For this study, we manually constructed snippets for results of queries on a popular e-commerce website. The snippets were constructed for first 15 results of each query and the snippet size k was fixed to k=6. An attribute that appears in 5 or fewer snippets can appear in at-most as many positions, whereas an attribute that appears in 6 or more (up to 15) result snippets can appear in between 1 and 6 positions, giving a total of 75 $\big(\sum_{i=[1,5]} i + 10 \times 6\big)$ possible configurations of an attribute in a result-set snippet. Instead of checking for all 75 configurations, we test on a subset consisting of 16 configurations, as shown in Table 1.

In particular, we chose five different values for the number of occurrences $nOccur(A_i)$ of attribute A_i, and for each value, we consider a number of positions $nPos(A_i)$ of A_i according to Table 1. For example, for Table 1 entry $(nOccur, nPos) = (8,2)$ corresponds to a snippet in which a particular attribute (e.g. *Price*) appears in 8 results and in 2 (vertically aligned) positions. The user is asked a single question about a particular attribute in the result set. These questions are designed to gauge the overall comprehension of the attribute. A sample question for the task of query '*acer laptop 3gb*' (Figure 1), might be '*Which product has the maximum Price?*' For each task, we measure the time taken to answer the question correctly, which estimates the value of $f(.)$.

3.3 User Study Results

We deployed the user study on Amazon Mechanical Turk and collected 83 valid responses after discounting users who abandoned the survey mid-way or took multiple attempts to answer. From these, we eliminated outliers, that is, responses that took unreasonably long. In particular, we removed entries with response times that were more than two standard deviations from the mean, leaving 57 responses on average, per task.

Figure 2 shows the response times of users to answer the question for different configurations of attribute arrangement. The first row of Figure 2 (2a-d) shows the plot of response times for differing number of occurrences of an attribute, while keeping the number of attribute positions in snippets ($nPos$), fixed. We observe that for a given number of positions, the mean response time does not vary significantly based on the number of occurrences ($nOccur$). For example, Figure 2b shows that when the number of positions of an attribute is fixed to 2 (second line, excluding heading, in Table 1), the mean response times for 4, 8, 12 and 15 occurrences of the attribute were 44.4, 42.9, 41.1 and 43 seconds, respectively, indicating that the response times, and therefore *effort, does not depend much on the number of occurrences of the attribute*. Intuitively, the reason is that once the user locates the position, it is fast to make a vertical scan to check the values of this attribute in the other snippets. This observation is supported by Figures 2a-d and also by statistical hypothesis tests for equivalence, as shown below.

Classical hypothesis testing procedures only provide a way to check non-equivalence. Since we want to test for equivalence of mean response times, we used the two one-sided testing (TOST) procedure, developed by Schuirmann et al. [14], which is widely used to test the bioequivalences of two drug formulations.

A TOST to test equivalences of means μ_1, μ_2 of two populations P_1, P_2 is formulated as follows:

$$H_O = \{(\mu_1 - \mu_2) \leq \theta_L \ or\ (\mu_1 - \mu_2) \geq \theta_U\}$$

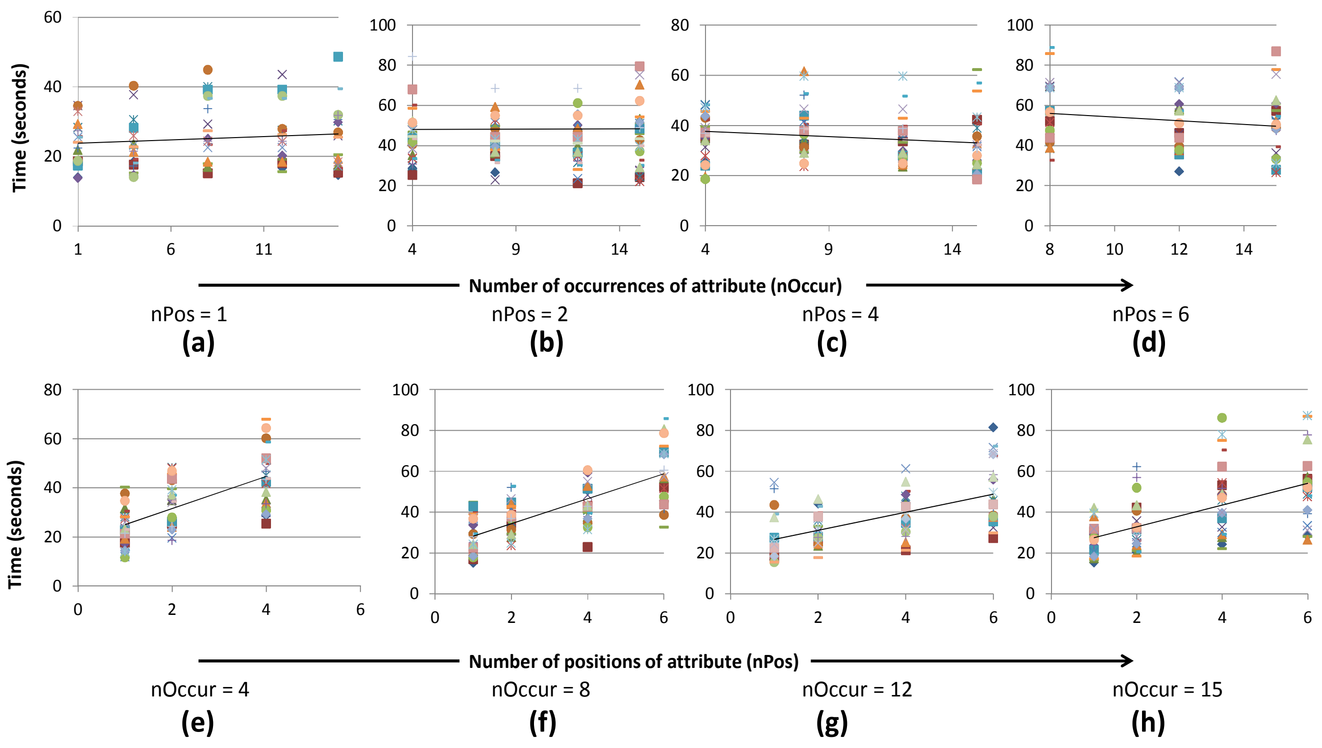


Figure 2. Comprehension Cost User Study Results

$H_A = \{\theta_L \le (\mu_1 - \mu_2) \le \theta_U\}$

where θ_L and θ_R are the upper and lower equivalence limits, respectively, which are defined by the test designer.

This test works by establishing a $100 \cdot (1 - 2\alpha)\%$ confidence interval (CI) for $\mu_1 - \mu_2$ and rejecting the null hypothesis H_O in favor of alternative hypothesis H_A if the CI falls within the equivalence limit $[\theta_L, \theta_U]$. We performed pairwise TOST statistical tests for each pair of mean response times for different number of occurrences ($nOccur$) for a given $nPos$ with an equivalence limit of ± 5 seconds and found that CI was within the limit when tested with $\alpha = 0.05$.

Of course, the above observation assumes that the number of snippets is reasonably small (15 in our experiment), which is the case in practice, given that the result-set snippet must fit in the screen. On the other-hand, as shown in Figures 2e-h, the number of positions of an attribute does affect navigation cost. This is because, the user has to expend more effort in navigating the result and check for each position of the snippet and look for a particular attribute. In particular, we see that the user time increases linearly with the number of positions.

We summarize our finding as follows. For a result-set snippet:

Observation 1: The comprehension cost does not depend on the number of occurrences of an attribute.

Observation 2: The comprehension cost increases linearly with the number of different positions of the attribute.

Therefore, the per-attribute cost function $f(.)$ can now be expressed as follows:

$$f(nPos(A_i)) = a \cdot nPos(A_i) + b \qquad (5)$$

To compute a and b we fit the response time against the number of positions into a linear function and obtained the following function, where the unit is seconds.

$$f(nPos(A_i)) = 5.17 \cdot nPos(A_i) + 22 \qquad (6)$$

We also experimented with higher order functions, but observed that they did not fit well with the data, which confirms our initial linearity observation. Note that the particular values for a and b depend on the nature of the result-set snippet, and particularly on factors like the number of snippets (15 in our experiment) and the comprehension difficulty of the attributes and values (see discussion in Section 3.1).

4. INFORMATIVENESS

In this section we present a set of factors that make a result-set snippet informative. In the example of Figure 1, it is useful to show the *Price* attribute since a user at an e-commerce website is typically very interested in the price of the product. The importance of an attribute in a result-set is subjective and depends on factors such as user preferences, global (result-set-independent) attribute importance or the distribution of attribute values in the result-set. Furthermore, informativeness could be defined at the attribute value level, instead of the attribute level. This is particularly desirable when different values have different importance for the user. For example, $Special\ Offer = true$ is more important than the $Special\ Offer = false$ value since it is advantageous to display a special offer to the user, if it is available for the given product.

There is no previous work defining the informativeness (or usefulness) of tabular snippets. For that, we borrow ideas from works on faceted navigation [5, 15], results diversity [16, 17, 9, 10], text snippets [24, 25] and XML snippets [7, 18]. These works define desirable principles for useful attributes or snippets, but do not provide a quantitative measure to compare the usefulness of two snippets. *Note that this section should not be viewed as a key contribution of our work, but is included for completeness.*

We introduce the function $I(.) \rightarrow \mathbb{R}^+$ to quantify the informativeness, and specifically two variants:

- $I(A_i, R)$: Informativeness of attribute A_i in result-set R
- $I(A_i, R, v)$: Informativeness of value v of A_i in R.

Some of the attribute usefulness factors that have been proposed in previous work are:

- *Distinguishability*: snippets should show the differences between results [8, 15, 21].
- *Diversity*: showing a variety of attributes gives to the user a broader view of the results [10, 17, 22, 23].
- *Importance*: show attributes that are more important in the result-set than in the whole database [7, 8, 20].

Without loss of generality, we assume that $I(.)$ assigns higher values to more informative attributes, i.e., if $I(A_i, R) > I(A_j, R)$, then it is preferable to include A_i in the result-set snippet instead of A_j. Next we present concrete ways to quantify $I(A_i, R)$ and $I(A_i, R, v)$, which use some of the proposed factors, and we use in our experiments in Section 7.

We define attribute level informativeness $I(A_i, R)$ as the complement of *indistinguishability* (INDG) [15], i.e., the maximum possible indistinguishability minus the attribute's indistinguishability:

$$I(A_i, R) = \frac{N(A_i, R)(N(A_i, R) - 1)}{2} - INDG(A_i, R) \quad (7)$$

where $N(A_i, R)$ is the number of times A_i appears in R and $INDG(A_i, R)$ is the *indistinguishability* score defined as:

$$INDG(A_i, R) = \sum_{a \in ADom(A_i, R)} \frac{|D(a, R)|(|D(a, R)| - 1)}{2}$$

where, $ADom(A_i, R)$ is the active domain of A_i in the result-set R and $|D(a, R)|$ is the number of times a value $a \in ADom(A_i, R)$ appears in R. For example, in Figure 1a, *I(Brand,R)=6(6-1)/2 – 3(3-1)/2 (Acer)- 0(Seagate) -0(Kingston) -0(Corsair)=12.*

Alternatively, we could use the entropy of these values or a user-specified global importance of the attributes.

For the value-specific informativeness, we adopt the dominance score from [7]:

$$I(A_i, R, v) = \frac{N(v, A_i, R)}{N(A_i, R)/|ADom(A_i, R)|} \quad (8)$$

where $N(v, A_i, R)$ is the number of times that v appears for attribute A_i in result-set R and $ADom(A_i, R)$ is the domain size of A_i in R and $N(A_i, R)$ is the number of times A_i appears in R.

Next, we define the total informativeness of result-set snippet $S(R, k)$ as the sum of the informativeness of its attributes:

$$Inform(S, R) = \sum_{A_i \in S} I(A_i, R) \quad (9)$$

For example in Figure 1b, $Inform(S, R) = 6 \cdot I(Brand, R) + 2 \cdot I(Capacity, R) + 3 \cdot I(Display, R) = 6 \cdot 3 + 2 \cdot 3 + 3 \cdot 4 = 36$.

The above formula assumes that the scores of attributes are independent from each other, which is a reasonable simplifying assumption, but clearly not true for all informative definitions.

If we also consider the attribute value-level informativeness $I(A_i, R, v)$, then the above formula can be rewritten as:

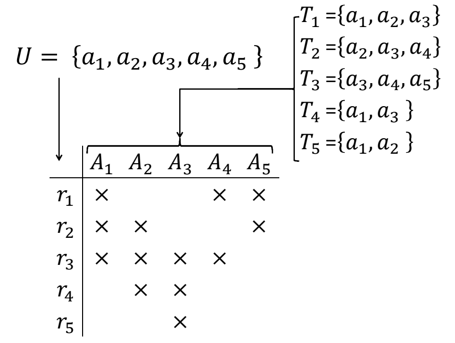
$$Inform(S, R) = \sum_{(A_i = v) \in S} I(A_i, R)\, I(A_i, R, v) \quad (10)$$

where for each condition $(A_i = v)$ in a snippet we multiply the informativeness of the value with that of the attribute.

5. COMPLEXITY RESULTS

In this section we study the complexity of the Snippet Construction Problem. We consider a simplified version of the Snippet Construction Problem, termed Fixed Snippet Construction (FSC), where the comprehension cost is the number of attribute positions. The key simplification in FSC is that it does not try to maximize informativeness. By showing that FSC is NP-hard, we also show that Snippet Construction Problem is NP-hard.

FSC Problem: Given a result-set R, construct a result-set snippet $S(R)$ (FSC has no snippet size constraint k), such that the comprehension cost is up to L and each snippet in $S(R)$ is non-empty. The comprehension cost for an attribute is the number of positions it appears in, and the comprehension cost of $S(R)$ is the sum over all attributes in $S(R)$. The informativeness is constant.



$U = \{a_1, a_2, a_3, a_4, a_5\}$
$T_1 = \{a_1, a_2, a_3\}$
$T_2 = \{a_2, a_3, a_4\}$
$T_3 = \{a_3, a_4, a_5\}$
$T_4 = \{a_1, a_3\}$
$T_5 = \{a_1, a_2\}$
A_1 A_2 A_3 A_4 A_5
r_1 × × ×
r_2 × × ×
r_3 × × × ×
r_4 × ×
r_5 ×

Figure 3. Reduction of Set Cover to the Fixed Snippet Construction (FSC) Problem

Theorem 1: FSC is NP-Complete.

Proof: The problem is obviously in NP. Given a result-set snippet $S(R)$, it is easy to verify that *S(R)* has comprehension cost L. To prove that the problem is NP-Complete, we reduce the Set Cover Problem (SCP) to FSC.

SCP: *Given a set of elements* $U = \{a_1, \ldots, a_n\}$ *and a set of subsets* $T = \{T_1, \ldots, T_m\}$ *of U find a subset* $T' \subseteq T$ *of size at most Z, such that* $\bigcup_{T_i \in T'} T_i = U$.

Given an arbitrary instance (U, T) of SCP, we construct an instance of FSC as follows. For each $a \in U$, create a result r_a. For each $T_i \in T$, create an attribute A_i, and add this attribute to all results r_a whose corresponding element a is in T_i. Figure 3 shows this reduction. Recall that results are heterogeneous, that is, they have different attributes. Finally, we map *Z=L*.

We now show that this mapping is indeed a reduction. A solution $S(R)$ to FSC is mapped to a solution T' to SCP by including to T' the subsets that correspond to the attributes in $S(R)$. Note that when we add an attribute A_i to $S(R)$, we simply add it to the snippets of all results that contain A_i since it does not increase the comprehension cost and there is no limit *k* on the size of a snippet. A solution to FSC is a solution to SCP because every result is non-empty, which means that every element in the universe in SCP is selected at least once. The comprehension cost of $S(R)$, which is the number of attributes in $S(R)$, is $|T'|$. Similarly, we can show the solution mapping in the other direction.

6. SNIPPET CONSTRUCTION ALGORITHMS

Challenges: Due to the intractability result of Theorem 1, in this section we propose efficient approximate algorithms for the

Snippet Construction problem. Intuitively, there are two sources of intractability regarding the comprehension cost:

(a) how to select which attributes to display in each snippet of the result-set snippet, and

(b) how to arrange them, i.e., assign positions.

To minimize the comprehension cost, we want to select common attributes across the snippets and assign them the same position, as in Figure 1d. However, we must also consider informativeness, which further complicates computation. Note that the informativeness contribution of an attribute or value in the result-set snippet *S* does not depend on the other attributes or values in *S*, according to the formulas presented in Section 4, which is clearly not true for the comprehension cost. Hence, the latter is the main complexity source that our algorithms must tackle.

Algorithm Overview: To create an efficient approximate algorithm, we carefully relax both intractability sources listed above. The result-set snippet construction algorithm, presented in Figure 4, iteratively constructs a result-set snippet $S(R,k)$ by greedily evaluating and adding one attribute at a time. The selected attribute is placed in the minimum number of positions possible in the partially constructed result-set snippet. We can view the result-set snippet as an initially empty $n \times k$ matrix. At each iteration, we select an attribute A_i and add it to the row of each result *r* that contains A_i, i.e., $A_i \in A^r$. We continue until either the matrix is full or adding an attribute decreases the goodness of the result-set snippet, as computed by Equation 2. The algorithm works by maintaining a $pool$ of candidate attributes that can be added to the snippet $S(R,k)$ along with auxiliary information about which results and the number of positions the attribute can be placed in the snippet matrix and a *heuristic* goodness score based on number of positions an attribute can occupy in $S(R,k)$. The attributes in the $pool$ are processed in the decreasing order of score and the remaining entries in the pool are updated to reflect this addition.

Algorithm Details: As a first step, the algorithm initializes a $pool$ of candidates (line 2). Each attribute A_i in the result-set R is represented in the pool by an entry e_i of the form $e_i{:}< A_i, nOccur, nPos, score >$ which includes the number of times $(nOccur)$ and the number of positions $(nPos)$ the attribute will appear in the final result-set snippet S. The entry also stores the $score$ as defined by Equation 2, which is computed by assuming that the snippet consists solely of the given attribute placed in the given number of positions $(nPos)$. Of course, the algorithm also has knowledge of which results in R contain which attributes.

The $pool$ is implemented as a priority queue arranged by decreasing $score$. The $pool$ is initialized (lines16-20) with entries for all attributes in R in their most optimal arrangement, i.e., assuming that all values of an attribute are added to the snippet in perfect alignment $(nPos = 1)$. Next, in lines 3-14 the snippet is built iteratively by processing attributes in decreasing order of $score$. At each step, the attribute with the maximum score is chosen (line 7) and added (line 13) to the result-set snippet with the configuration (places and positions) dictated by the entry.

Given that an attribute is added independently of others, it is possible that the entry being processed cannot be added to the snippet in the configuration dictated by the entry, in $nPos$ positions occupying $nOccur$ spots in the snippets. This situation arises when potential spots are filled up by other attributes in previous iterations. For instance, attribute A_5 may be part of the 2nd and 4th result of the result-set, but the result-set snippet matrix does not have any position (column) for which both the cell of the 2nd and 4th rows are free. This situation is handled in lines 5-6, where this incompatibility is checked and the $pool$ is recomputed (lines 21-26) by adjusting the positions and places the remaining attributes in the pool can occupy, given the snippet $S(R,k)$ computed thus far.

The algorithm also maintains the *global* informativeness and comprehension cost of the partially constructed result-set snippet S. It is possible that adding an attribute would decrease the overall goodness of a snippet. For example, if the attribute being added has a very low informativeness and it is being added to many different positions, then the overall informativeness of the snippet can potentially decrease. To avoid this, the algorithm checks (lines 11 & 12) to see if adding the attribute would decrease the overall score. If the global score decreases, not only is the attribute not added, but the computation stops since any attribute that is added in future would not increase the score.

Algorithm: *SnippetConstructionAlgorithm*

Input: Result-set R, snippet size bound k and trade-off parameter λ.

Output: Snippet $S(R,k)$ of R with size bound k

```
1. $prevScore = 0,\ inform = 0,\ compCost = 0$
2. $pool < A_i, nOccur, nPos, score > \leftarrow$ initPool()
3. while $(pool.size > 0$ and $S$ is not $full)$
4.     $e \leftarrow pool.peekMax()$
5.     if $(incompatible(e,S))$
6.         recomputePool();
7.     $e \leftarrow pool.removeMax()$;
8.     $prevScore \leftarrow \frac{inform^{\lambda}}{compCost^{1-\lambda}}$
9.     $inform \leftarrow inform + Inform(e.nOccur)$
10.        $compCost \leftarrow compCost + Compreh(e.nPos)$
11.    if $\left(prevScore < \frac{inform^{\lambda}}{compCost^{1-\lambda}}\right)$
12.        stop and return $S$.
13. $add(e,S)$ // add $e.nOccur$ instances of $e.A_i$ to $S$ at
               // $e.nPos$ positions
14. end while
15. return $S$.
```

Procedure: *initPool*

Input: Result-set R and trade-off parameter λ.

Output: The $pool$ of candidate attributes to add to snippets

```
16. foreach $A_i \in attributes(R)$
17.     $nOccur \leftarrow numResults(A_i, R)$
18.     $e = < A_i, nOccur, 1, \frac{Inform(A_i)^{\lambda}}{Compreh(A_i)^{1-\lambda}} >$
19.     $pool.add(e)$
20. endfor
```

Procedure: *recomputePool*

Input: A $pool$ of candidates, the partial result-set snippet S.

Output: The $pool$ of candidate attributes to add to snippets

```
21. foreach $e \in pool$
22.   if $(incompatible(e))$
23.       while $(incompatible(e))$
24.           Alternately $e.nPos$ ++ or $e.nOccur$--
25.           $e = < A_i, nOccur, nPos, \frac{Inform(A)^{\lambda}}{Compreh(A)^{1-\lambda}} >$
26. endfor
```

Figure 4. Result-set Snippet Construction Algorithm

The adjustment (line 24) works by increasing the number of positions $(e.nPos)$ or decreasing number of results that it can be placed in $(e.nOccur)$. For example, in the case of A_5 above, the algorithm might decide to place A_5 in the snippet of only one of the result (2nd or 4th), depending on availability or it might choose to place them in two different positions (columns). The algorithm prioritizes informativeness over comprehension cost, therefore attempts to place the attribute in multiple positions (by increasing $nPos$) before decreasing informativeness. The score of the attribute is recomputed (line 25) which might result in a change of position in the score ordered $pool$.

To check if a given configuration (entry) of an attribute is compatible, the $incompatible(e, S)$ method (not shown in Figure 4) scans the positions (columns) of the partially constructed snippet $(S(R,k))$ in the decreasing order of number of cells available for coverage of the attribute in results. For example, if an attribute A_6 appears in 10 of (say) 15 results $(\text{i.e.}\ e_6.nOccur = 10)$, and the configuration dictates that the attribute should be placed in 2 positions $(e_6.nPos = 2)$, then the $incompatible(e_6, S)$ method looks for two positions (columns) with empty cells that would cover 10 occurrences by scanning positions (columns) in the snippet matrix in decreasing order of number of empty cells. The rationale for doing so is to fit the given attribute in the minimum number of possible positions.

Complexity: Let N_a be the number of attributes in R. The worst case complexity of the *SnippetsConstructionAlgorithm* is $O(kN_a^2)$ since the algorithm (lines 3-13) considers a $pool$ of N_a attributes to be placed in k positions, giving a complexity of kN_a. Furthermore, in each iteration the $pool$ can be recomputed, to give the overall (worst-case) running time. However, this worst-case running time is misleading as the pool is rarely recomputed, especially in the first few iterations. We further employ efficient bit-vector manipulation to check for incompatibility making the cost of $incompatible$ negligible.

***SnippetConstructionAlgorithm* with Value Informativeness:** The snippet construction algorithm of Figure 4, which uses attribute-based informativeness (Equation 9) can be easily adapted to consider attribute-value based informativeness (Equation 10). Instead of using per placement attribute informativeness (lines 18 and 25), we sum up the informativeness of individual values of valid configuration of the attribute value, while computing the scores. The rest of the algorithm remains unchanged.

Table 2. Query Workload

	Query	# Categories	# Attributes
Q1	toshiba laptop	6	39
Q2	dell laptop 3GB	7	43
Q3	dell printers	4	40
Q4	Asus laptops	7	40
Q5	hard drive	7	42
Q6	dell Intel	6	55
Q7	seagate drive	7	40
Q8	dell 13.3	6	51
Q9	hp printer	7	42
Q10	seagate 1TB	4	52

7. EXPERIMENTAL EVALUATION

In Section 7.1, we describe the experiment setup including the datasets used, query workload and the comparison baselines. We present the results of the experiments in Section 7.2 and show that snippets constructed using our methods effectively balance comprehension cost and informativeness. Section 7.3 measures the time requirements of our heuristic algorithm and shows that it adds an insignificant overhead to overall query processing.

7.1 Experimental Setup

Dataset and Query Workload: We evaluate our approach on a subset of products data from a popular e-commerce website. The dataset consists of diverse products *types* such as *Computers (Desktops, Laptops etc.)*, *Printers (InkJet, LaserJet etc.)* among many others. In total we extracted 63,126 products from 52 categories (types) and 150 unique attributes distributed amongst the different types. The queries used in the evaluation are shown in Table 2. Table 2 also shows some characteristics of the result-set of each query, namely, the number of categories (types) and the total number of unique attributes in query results.

Baselines: We compare our approach with the following two commonly used methods to construct snippets:

Baseline 1 (Fixed-Schema): In this snippet construction method, the schema of the result-set is fixed by choosing the most commonly occurring attributes in results. In a heterogeneous result-set, a result-set snippet would have many empty $(null)$ values, since a result may not have a value for one or more selected attributes and therefore be less informative. However, these snippets have a low comprehension cost, since all displayed attributes are aligned.

Baseline 2 (Popular-attributes): This method chooses the most informative attributes *for each result*, as its snippet. The informativeness of an attribute is computed based on the result-set using Equation 9. The k selected attributes from each result are displayed in a tabular format and are ordered by decreasing informativeness scores. These snippets are highly informative, since they always select the maximum possible k informative attributes, for each snippet. However, the comprehension cost would be high, due to partial non-alignment of attributes across snippets and inclusion of *superfluous* or unnecessary attributes.

Evaluation Methodology: For each query, we construct snippets of one page of the results, i.e. 15 results, using the algorithm described in Section 6 and for the aforementioned baselines. The snippets size k was set to 6 in all our experiments. For snippets constructed using each method, we compute the total comprehension cost and attribute informativeness using Equations 5 and 7 & 9 respectively and report the absolute numbers. The results for attribute value informativeness are similar and are omitted due to space limitations. We also report the combined cost $\mathcal{F}$, using Equation 2. All experiments were performed by setting $\lambda = 0.5$, i.e. by giving equal weights to informativeness and comprehension factors.

7.2 Results

The resulting goodness scores for the snippets of the 10 queries used in the evaluation are shown in Figure 5. The goodness scores for our snippet construction algorithm are much better on average as compared to the other two baselines, with an improvement of nearly 27% over Fixed-Schema and 32% over Popular-Attributes approach. Both the Fixed-Schema and Popular-Attributes baseline have similar scores for these queries, since Fixed-Schema approach minimizes comprehension cost while Popular-Attributes maximizes informativeness. Our approach balances both these factors, thereby achieving higher scores.

Figure 6 shows the overall informativeness of the snippets constructed using the two baselines and our approach. As expected, the informativeness of Fixed-Schema approach is uniformly lower than other two approaches. This is because, by fixing the schema, a number of places in the snippets remain empty (as in Figure 1b) or less informative attributes get selected. In contrast, the informativeness of snippets constructed by the Popular-Attributes baseline is better than other two approaches since each position in the snippet is occupied by highly informative attributes. For our approach, the informativeness is lower than Popular-Attributes, by about 22%. This is expected, since we sacrifice informativeness in an effort to make a snippet more comprehensible. By selecting attributes based on comprehension cost, in addition to informativeness, results in selection of some attributes that have low informativeness. Also, some positions in the snippets might remain unoccupied since adding additional attributes might result in degrading of the overall *goodness* score of the snippet.

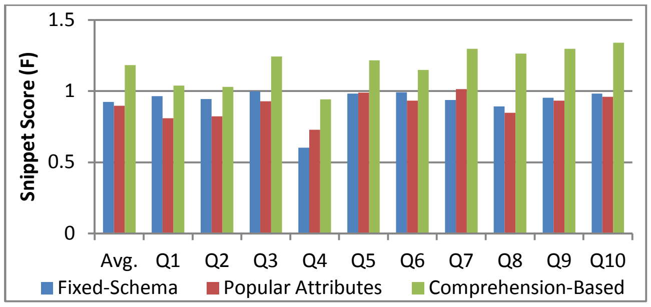


Figure 5. Goodness Score ($\mathcal{F}$) of Result-set Snippets

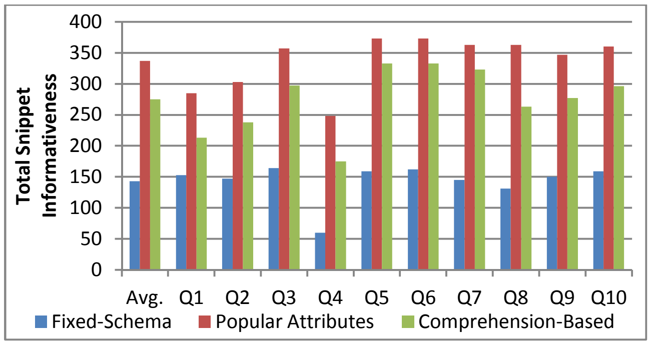


Figure 6. Total Informativeness of Attributes in Snippets

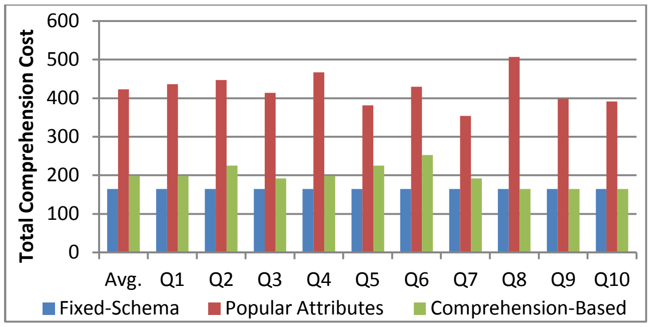


Figure 7. Total Result-set Snippet Comprehension Cost

The loss in informativeness is more than made up (113% improvement, on average) when we consider the comprehension cost, in Figure 7, of our snippets to those constructed by the Popular-Attributes approach. The Popular-Attributes baseline does not consider comprehension cost and therefore arranges attributes in *jagged* or non-aligned order across results resulting in a high comprehension cost. Additionally, by selecting attributes independently for each result, the number of attributes that are selected across snippets is high, thereby adding a huge comprehension cost overhead.

The snippets constructed using the Fixed-Schema approach have a fixed comprehension cost (Figure 7). This is the lowest possible cost, given that all positions in the snippet are occupied. Our approach, which balances comprehension and informativeness, constructs snippets with higher comprehension cost (by 20%) but the resulting snippets are also highly informative (Figure 6).

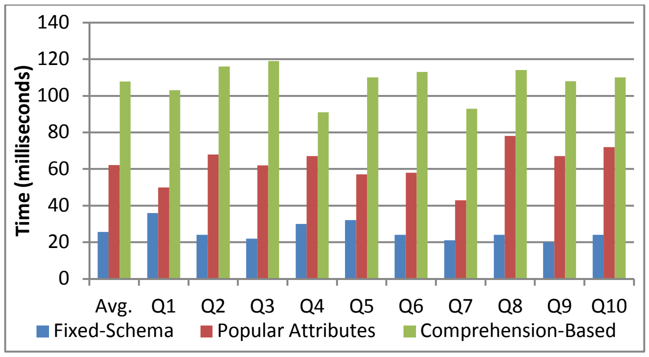


Figure 8. Snippet Construction Algorithm Performance

7.3 Performance Experiments

Figure 8 compares the execution times of our approach with those of the two baselines. The Fixed-Schema approach, whose schema is fixed beforehand, takes almost negligible amount of time (26ms) on average. Our algorithm takes much longer, 107ms on average, which is considerably higher than Popular-Attributes (avg. 63ms), but is still fast and adds a small overhead to the search and retrieval process since the time to execute the query and retrieve results is considerably higher.

8. RELATED WORK

Snippets of text documents: There has been much work on snippets construction for Web documents. Earlier work relied on query-independent document summarization techniques [24, 25]. More recently, query-specific snippet generation techniques have been proposed [26]. However, none of these works consider the problem of comprehension cost across snippets, since each snippet is just a list of sentences. We could also tailor our approach to make snippets query-specific by incorporating the query relevance into the informativeness model.

Faceted Search on Structured Data: Faceted search employed by most e-commerce websites (Amazon, eBay) typically supports predefined top-down navigation on the concept hierarchy, where all the attributes of the currently selected concept are also displayed. Recent works [6, 15] have studied the problem of what attributes to display to minimize the user effort, but operate on flat relations of products without any classification. BioNav [5] presents the bibliographic results of queries on PubMed on an ontological hierarchy and allows users to effectively navigate on that hierarchy. However, attributes are not considered and each publication is viewed as a result hung on a leaf of the hierarchy.

Search Results Comprehension: *Comprehension*, as applied to designing data-driven user interfaces is a subjective measure that falls into the realm of cognitive psychology [27] and Computer Human Interaction (HCI) and focuses on identifying design features of user interfaces and results presentation that maximize the efficiency of user in understanding the set of results or the interface in question. We use the methods and techniques from cognitive psychology and computer human interaction to design user-studies to develop our comprehension cost model and here we discuss works that are related to the goals of our user-study.

Dalal et al. [12] propose a design of website's home pages along several *theoretical* guidelines and conduct user studies to measure the comprehension of web-site home pages along three dimensions – accuracy, speed and perceived comprehension and conclude that pages designed with their theorized guidelines achieve better cognition. We also theorize a model of comprehension (Section 3) and we use the user study to parameterize the model (define the function $Compreh(.)$). Some recent works [11, 13] study the effect of layout of data elements on their cognition. Our study focuses on measuring the effect of placement of snippet constituents near each other. [11, 13] conduct eye-movement studies of pieces of text to conclude with a direct correlation between placement of target text and comprehension, measured as answers to questions. We follow a similar mechanism, but instead use time spent in reading and answering questions as measure of comprehension.

9. CONCLUSIONS

In this paper, we introduced the problem of incorporating the user comprehension cost into the construction of snippets for structured data. In particular, we defined the framework and identified the criteria for constructing snippets that minimize the effort required by users in comprehending the set of results. We presented a complexity analysis of the problem and efficient algorithms to construct snippets that minimize the comprehension cost, while maximizing the informativeness of the snippets. Our results are supported by user studies and quantitative experiments.

10. ACKNOWLEDGMENTS

This project was supported in part by National Science Foundation grants IIS-1216032 and IIS-1216007.

11. REFERENCES

[1] Balmin, A., Hristidis, V. and Papakonstantinou, Y. Objectrank: authority-based keyword search in databases. In Proceedings of VLDB (2004).

[2] Guo, L., Shao, F., Botev, C. and Shanmugasundaram, J. XRANK: ranked keyword search over XML documents. In Proceedings of SIGMOD (2003).

[3] He, H., Wang, H., Yang, J. and Yu, P. S. BLINKS: ranked keyword searches on graphs. In Proceedings of SIGMOD (2007).

[4] Chakrabarti, K., Chaudhuri, S. and Hwang, S.-w. Automatic categorization of query results. In Proceedings of SIGMOD (2004).

[5] Kashyap, A., Hristidis, V. and Petropoulos, M. FACeTOR: cost-driven exploration of faceted query results. In Proceedings of CIKM (2010).

[6] Kashyap, A., Hristidis, V., Petropoulos, M. and Tavoulari, S. BioNav: Effective Navigation on Query Results of Biomedical Databases. In Proceedings of ICDE, 2009.

[7] Huang, Y., Liu, Z. and Chen, Y. Query biased snippet generation in XML search. In Proceedings of SIGMOD (2008).

[8] Liu, Z., Sun, P. and Chen, Y. Structured search result differentiation. Proceedings of VLDB, 2, 1 (2009), 313-324.

[9] Marcos, R. V., Razente, H. L., Barioni, M. C. N., Hadjieleftheriou, M., Srivastava, D., Jr., C. T. and Tsotras, V. J. On query result diversification. In Proceedings of ICDE (2011).

[10] Smyth, B. and McClave, P. Similarity vs. Diversity. In Proceedings of the International Conference on Case-Based Reasoning. (2001).

[11] Bicknell, K. and Levy, R. Rational eye movements in reading combining uncertainty about previous words with contextual probability. In Proceedings of the Conference of the Cognitive Science Society.(2010).

[12] Dalal, N. P., Quible, Z. and Wyatt, K. Cognitive design of home pages: an experimental study of comprehension on the World Wide Web. Information Processing & Management, 36, 4 (2000), 607-621.

[13] Dubey, A., Keller, F. and Sturt, P. The Effect of Phonological Parallelism in Coordination: Evidence from Eye-tracking. In Proceedings of European Cognitive Science Conference (2007).

[14] Schuirmann, D. J. A comparison of the two one-sided tests procedure and the power approach for assessing the equivalence of average bioavailability. J Pharmacokinet Biopharm, 15, 6 (1987), 657-680.

[15] Roy, S. B., Wang, H., Das, G., Nambiar, U. and Mohania, M. Minimum-effort driven dynamic faceted search in structured databases. In Proceedings of CIKM (2008).

[16] Agrawal, R., Gollapudi, S., Halverson, A. and Ieong, S. Diversifying search results. In Proceedings WSDM (2009).

[17] Clarke, C. L. A., Kolla, M., Cormack, G. V., Vechtomova, O., Ashkan, A., Battcher, S. and MacKinnon, I. Novelty and diversity in information retrieval evaluation. In Proceedings of SIGIR (2008).

[18] Chen, H. and Karger, D. R. Less is more: probabilistic models for retrieving fewer relevant documents. In Proceedings of SIGIR (2006).

[19] Agrawal, R., Gollapudi, S., Halverson, A. and Ieong, S. Diversifying search results. In Proceedings of WSDM (2009).

[20] Das, G., Hristidis, V., Kapoor, N. and Sudarshan, S. Ordering the attributes of query results. In Proceedings of SIGMOD (2006).

[21] Roy, S. B., Amer-Yahia, S., Chawla, A., Das, G. and Yu, C. Constructing and exploring composite items. In Proceedings of SIGMOD (2010).

[22] Carterette, B. An Analysis of NP-Completeness in Novelty and Diversity Ranking. In Proceedings of ICTIR (2009).

[23] Jain, A., Sarda, P. and Haritsa, J. Providing Diversity in K-Nearest Neighbor Query Results. Advances in Knowledge Discovery and Data Mining, Springer Berlin / Heidelberg, (2004).

[24] Turpin, A., Tsegay, Y., Hawking, D. and Williams, H. E. Fast generation of result snippets in web search. In Proceedings of SIGIR (2007).

[25] White, R. W., Ruthven, I. and Jose, J. M. Finding relevant documents using top ranking sentences: an evaluation of two alternative schemes. In Proceedings of SIGIR (2002).

[26] Varadarajan, R. and Hristidis, V. A system for query-specific document summarization. In Proceedings of CIKM (2006).

[27] Kitajima, M., Blackmon, M. H. and Polson, P. G. A Comprehension-based Model of Web Navigation and Its Application to Web Usability Analysis. Proceedings of HCI (2002).