FACeTOR: Cost-Driven Exploration of Faceted Query Results

ABSTRACT

Faceted navigation is being increasingly employed as an effective technique for exploring large query results on structured databases. This technique of mitigating the information overload problem leverages metadata of the query results to provide users with facet conditions that can be used to progressively refine the user's query and filter the query results. However, the number of facet conditions can be quite large, thereby increasing the burden on the user. In this paper, we propose the FACeTOR system that proposes a cost-based approach to faceted navigation. At each step of the navigation, the user is presented with a subset of all possible facet conditions that are selected based on a cost model of user navigation, such that the overall expected navigation cost is minimized and every result is guaranteed to be reachable by a facet condition. We prove that the problem of selecting the optimal facet conditions at each navigation step is NP-Hard, and subsequently present two intuitive heuristics employed by FACeTOR. Our user study at Amazon Mechanical Turk shows that FACeTOR reduces the user navigation time compared to the cutting edge commercial and academic faceted search algorithms. The user study also confirms the validity of our cost model. Finally, we performed an extensive experimental evaluation using two real datasets on the performance of the proposed algorithms. A prototype of FACeTOR is available at http://facetor.com.

1. INTRODUCTION

In recent years, there has been a tremendous increase in the number and size of databases published online, commonly referred to as the "deep web" [5], exposing a wide range of content including product catalogs (e.g. Amazon, eBay), bibliographies (e.g. DBLP, CiteSeer, PubMed), local businesses (e.g. Yelp) and many more. These databases are commonly queried using forms or keyword-based interfaces.

When users are not familiar with the content or the structure of the underlying database, or they are not experienced with sophisticated search interfaces, they issue queries that are exploratory in nature and may return a large number of results. In other cases, users often issue broad (underspecified) queries in fear of missing potentially useful results. As a consequence, users end up spending considerable effort browsing long lists of query results. This phenomenon, known as *information overload*, is a major hurdle in querying large databases.

Information overload has been tackled from two directions – ranking and categorization. There have been many recent works on ranking the database results for both keyword [2,16] and structured queries [7]. Ranking is effective when the assumptions used by the ranking function are aligned with the user preferences. Ranking may not perform well for exploratory queries, since it is hard to judge which result is better than the other when the query is broad. Moreover, no summary (grouping) of the query result is provided for the user to refine her query.

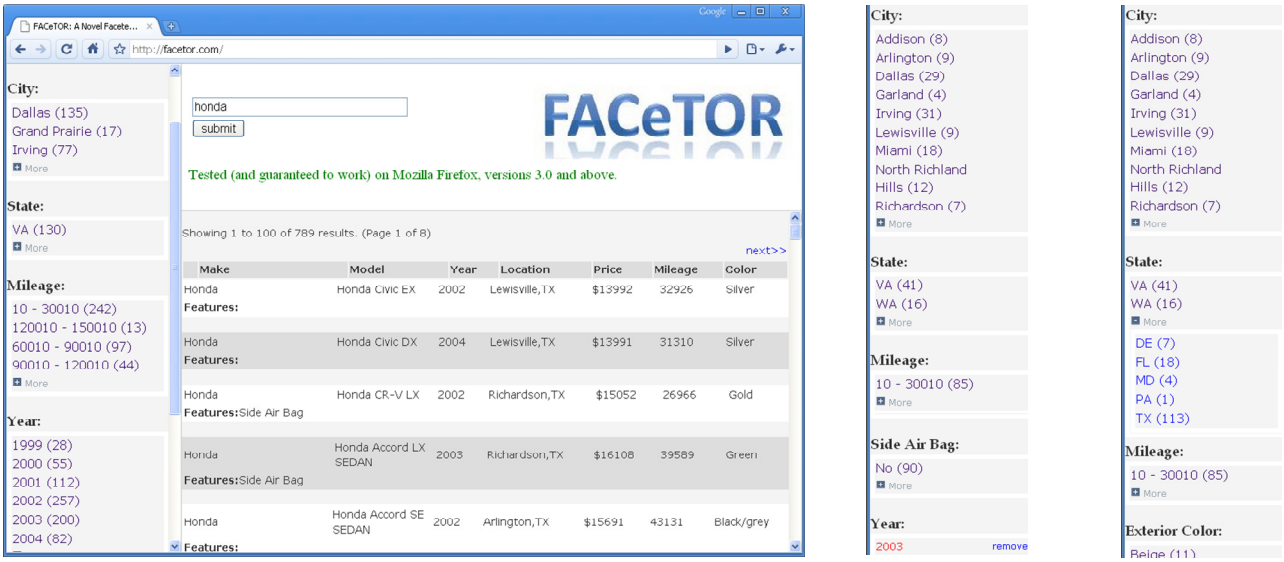
In categorization, query results are grouped based on hierarchies, keywords, tags, or attribute values. For instance, consider the MEDLINE database of biomedical citations [21], whose articles are tagged with terms from the MeSH concept hierarchy [19]. Categorization systems propose a method for users to effectively explore the large results by navigating the MeSH sub-hierarchy relevant to the particular query result [18]. Wider adoption of such hierarchical categorization systems is limited though as building these concept hierarchies requires an intense manual effort, and automatically assigning terms to tuples afterwards is not always a successful process [14].

A popular variant of categorization, which is the focus of this paper, is *faceted navigation* [10,22]. Here, the tuples in a query result are classified into multiple independent categories, or *facets*, instead of a single concept hierarchy. A facet comprises "a clearly defined, mutually exclusive, and collectively exhaustive aspect, property or characteristic of a class or specific subject" [24]. For an example car dataset, the result for keyword query "$honda$" shown in Fig. 1a is categorized based on $Year$, $City$ and $State$ facets, among others. Each facet is associated with a set of *facet conditions*, each of which appears in the number of tuples shown in parenthesis (cardinality). For instance, the $Year$ facet in Figure 1a is associated with the set $\{2000, 2001, \ldots\}$ of facet conditions. The user can narrow down or *refine* this result set by selecting a facet condition (e.g., $Year = 2003$) and clicking on it. The system then filters the query result to contain only the tuples satisfying the selected facet condition and displays the remaining facets conditions for the next navigation step, as shown in Figure 1b. User studies have showed that faceted navigation improves the ability of users to explore large query results and identify tuples of interest when compared to single concept hierarchies [27].

Faceted navigation has been studied extensively by the Information Retrieval community, where the challenge is to dynamically determine the keyword facets for a given set of documents, assign classifications to fragments, and then organize the documents using these classifications [11]. The drawback of these systems is the unpredictability and counter-intuitiveness of the resulting facets [14,15]. In contrast, faceted navigation is much more intuitive and predictable for structured databases, where each attribute is a facet describing a particular characteristic of the tuples in the dataset.

The following are some key concerns that need to be addressed to achieve effective faceted navigation when the number of facets and facet conditions are large:

1. Which facets and facet conditions should be suggested (displayed) to the user? For example, the query result of Figure 1a consists of 789 tuples that can be categorized using 41 facets and 234 facet conditions. Suggesting "familiar" facets and facet conditions would help the user make a refinement decision without requesting additional facet conditions (by clicking the "More" hyperlinks in Figure 1a).



(a) Initial Query Results and Suggested Facet Conditions (b) REFINE Action Effect (c) EXPAND Action Effect

Figure 1. The FACeTOR Interface

For example, most users are more familiar with the $Year$ facet than the $Mileage$ facet in Figure 1a. Most current solutions, including the ones employed by Amazon and eBay, try to address the facet conditions selection problem in an *ad hoc* manner by ranking the facet conditions using results cardinality or other hard-coded factors.

2. Which facet conditions will lead to the tuples of interest in fewer navigation steps? For example, the $Make = Honda$ facet condition has the highest cardinality for the query result in Figure 1a, but should not necessarily be suggested since it does not significantly prune the query results, that is, it does not help reduce the faceted navigation steps.

3. The overlap of the query results among the set of suggested faceted conditions is another critical concern, since a low overlap can reduce the facet conditions inspected and shorten the navigation. In Figure 1a, if most of the $City = Dallas$ cars were made in $Year = 2001$, it is not wise to suggest both facet conditions, because if the user chooses one of them in one navigation step, she would have to inspect the other in the next step anyway.

We present the FACeTOR system that takes a cost-based approach to selecting the best set of facet conditions to suggest to the user at each navigation step. These facet conditions are selected using an intuitive cost model that captures the expected cost of navigating a query result. At each navigation step, FACeTOR first computes the facet conditions applicable to the query result. However, instead of showing all of them or ranking them by an *ad hoc* function, FACeTOR suggests a subset of these facet conditions based on an intuitive user navigation cost model, which considers factors including the user's familiarity with the suggested conditions, their overlap, and the expected number of navigation steps. The suggested facet conditions are chosen such that they minimize the expected navigation cost until the tuples of interest are reached, although these are not known *a priori*.

Recent work on faceted navigation of database query results [6,22] has the following limitations, which we address in this paper. In both works, the navigation algorithm selects one facet (or possibly multiple ones [22]) and displays *all* its facet conditions to the user. Instead, we suggest a mix of facet conditions from several facets, that is, our algorithm operates at the facet condition level and not the facet level. Further, our cost model more closely estimates the actual user navigation cost. These improvements introduce novel algorithmic challenges, due to the explosion of the search space and the interactive time requirement of exploration systems. This paper makes the following contributions:

1. A complete framework for faceted navigation of structured query results (Section 2).
2. Intuitive navigation and cost models that closely resemble the actions taken by the user during faceted navigation of query results (Section 3).
3. The cost models introduced in Section 3 are necessarily probabilistic, given the uncertainty of user actions. We introduce these probability measures in the cost model (Section 3) and present methods of estimating these probabilities in Section 5.
4. A theoretical modeling and analysis of the problem of selecting the best facet conditions to suggest at each navigation step; we prove that this problem is NP-Hard given our navigation model (Section 4).
5. Two efficient and intuitive heuristics for the above problem (Section 6).
6. An extensive experimental evaluation with two real datasets showing that FACeTOR outperforms state of the art systems (Section 7).
7. A large-scale user study showing that FACeTOR decreases the user navigation time, which is proportional to the estimated navigation cost computed by our cost model (Section 8).

We discuss related work in Section 9 and conclude in Section 10.

2. FACETOR FRAMEWORK

The starting point of the FACeTOR framework is a *result set* that the user explores.

Definition 1 (Result Set) A result set is a relation R with schema $S_R = \{A_1, \ldots, A_m\}$. Each attribute $A_i \in S_R$ has an associated active domain $ADom(A_i, R)$ of un-interpreted constants, including the null value. □

The active domain $ADom(A_i, R)$ of an attribute A_i under R is defined as the set of all constants returned by the query over R projecting only A_i [1]. The initial result set R could be the whole database or more realistically, the result of a keyword query. In this work, we assume that the user first submits a keyword query (e.g., "$honda$" in Figure 1a). The result of this query forms the initial result set R, which the user explores. At each step of a faceted navigation, FACeTOR classifies the tuples of a result set R according to their *facets*. Each attribute $A_i \in S_R$ of R contributes a facet to the classification and, in turn, each facet contributes a set of conditions.

Definition 2 (Facet Condition): Given a result set R, a facet condition is an equality predicate $c: A_i = a_i$, where $A_i \in S_R$ and $a_i \in ADom(A_i, R)$. □

The set of all possible facet conditions for a result set R is denoted as $C(R)$.

Our running example considers a cars result set R whose tuples are classified by their $Year$, $City$, $Exterior\ Color$ and 37 more facets. As shown in Figure 1a, FACeTOR displays the name of each facet along with a list of facet conditions as hyperlinks, followed in parenthesis by the number of tuples in R satisfying the condition (cardinality).

When the user clicks on a hyperlink corresponding to a facet condition c_i, FACeTOR filters the result set R to the tuples that satisfy c_i, thus yielding a new result set $R_Q \subseteq R$, and the faceted navigation proceeds to the next step where R_Q is now being classified. FACeTOR captures the progression of the faceted navigation using a query Q, which is initially the identity query on the result set R. When the user clicks on a facet condition c_i, then the equality predicate is added conjunctively to Q, thus forming a refined query $Q \wedge c_i$.

For example, if on Figure 1a the user clicks on hyperlink 2003, FACeTOR executes the query $Q \wedge Year = 2003$ and filters the result set to the 200 tuples satisfying this facet condition. Subsequently, all facet conditions in $C(R_{Q \wedge Year=2003})$ are displayed, as shown in Figure 1b. At each navigation step, FACeTOR *suggests* only a subset C_S of all possible facet conditions in $C(R_Q)$.

Definition 3 (Suggested Conditions): For a result set R_Q, a set of facet conditions $C_S(R_Q) \subseteq C(R_Q)$, are suggested if $\bigcup_{c \in C_S(R_Q)}(R_{Q \wedge c}) = R_Q$, that is, every tuple in R_Q satisfies at least one suggested condition. □

In this work, we are interested in minimizing the overall expected navigation cost incurred by the user, by choosing the *best* set of suggested conditions for a given R_Q, without making any assumptions about the user's preference over the tuples in R_Q. The navigation cost is based on an intuitive model of user navigation described next.

Table 1. Symbol Reference

Symbol	Meaning
S_R	The schema of the initial result set with attributes $A_1, \ldots, A_m$
R	The initial result set.
Q	The query formulated during a faceted navigation.
R_Q	$R_Q \subseteq R$, the result of a query Q over R.
c	A facet condition of the form $A_i = a_j$.
$C(R_Q)$	All possible facet conditions for R_Q.
$C(A_i)$	All possible facet conditions for attribute A_i.
$C_S(R_Q)$	$C_S(R_Q) \subseteq C(R_Q)$, the suggested conditions given R_Q.
$P_{SR}(R_Q)$	The probability the user performs a SHOWRESULT action.
$P(c)$	The probability the user performs a REFINE action by a facet condition c.
$P_A(A_i)$	The probability the user prefers attribute A_i over all other attributes.

3. NAVIGATION AND COST MODELS

The faceted navigation model of FACeTOR is formally presented in Section 3.1 and forms the basis for the navigation cost model defined in Section 3.2. In Section 3.3, we give the intuition behind the navigation cost model and discuss how it affects the choice of facet conditions FACeTOR suggests to the user.

3.1 Faceted Navigation Model

At each faceted navigation step, FACeTOR displays to the user the set of suggested conditions $C_S(R_Q)$ for the current result set R_Q. The user then explores R_Q by examining all conditions in $C_S(R_Q)$ and proceeds to the next navigation step by performing one of the following actions:

1. **SHOWRESULT(R_Q):** The user examines all tuples in the result set R_Q. If in Figure 1a the user chooses to stop navigation and read all the results, she would have to read a total of 789 result tuples and 21 labels.
2. **REFINE(Q, c):** The user chooses a suggested condition $c \in C_S(R_Q)$ and refines query Q, that is, Q becomes $Q \wedge c$. The result of $REFINE(Q, c: year = 2003)$ is shown in Figure 1b. As a consequence of this action, the result set has now been narrowed down to 200 tuples and the new set of suggested conditions is available for this *refined* result set.
3. **EXPAND(A_i, R_Q)** : The user is dissatisfied with (rejects) *all* suggested conditions in $C_S(R_Q)$. Instead, she EXPANDs an attribute A_i by clicking on its "More" hyperlink, which reveals the remaining facet conditions for A_i in R_Q, and selects one of them to REFINE the query Q. EXPAND occurs when the user is not familiar with any of the suggested conditions. The effect of EXPAND on $State$ attribute is shown in Figure 1c, where the remaining facet conditions for $State$ are revealed. She then selects one of the facet conditions in $C(A_i) \backslash C_S(R_Q)$ and REFINEs the result set R_Q.

The formal model of user navigation is presented in Figure 2. It is a recursive procedure and is initially called on the entire result set R and the identity query Q, and terminates when the user finds all the tuples of interest, that is, when the user executes SHOWRESULT(R_Q). FACeTOR computes the result set R_Q and the suggested conditions $C_S(R_Q)$ at the beginning of each NAVIGATE step.

```
NAVIGATE(Q)
1   Choose one of the following:
2     SHOWRESULT(R_Q)
3     Examine all suggested conditions C_S(R_Q)
4       Choose one of the following:
5         REFINE(Q, c)
6           Q = Q ∧ c
7         EXPAND(A_i, R_Q)
8           Examine all remaining conditions in C(A_i)\C_S(R_Q)
9           Choose a condition c' ∈ (C(A_i)\C_S(R_Q))
10          Q ← Q ∧ c'
11      NAVIGATE(Q)
```

Figure 2. Faceted Navigation Model

3.2 Cost Model

The cost model measures the navigation cost incurred by the user when exploring a query result set R_Q, using the navigation model described in Section 3.1. The navigation cost is the sum of costs of the actions performed by the user, that is, examining suggested conditions, SHOWRESULT, REFINE and EXPAND actions.

The cost of examining all tuples in a result set R_Q, that is, the cost of SHOWRESULT(R_Q), is $|R_Q|$, and the cost of examining all suggested conditions is $|C_S(R_Q)|$.We assume that the REFINE and the EXPAND actions have a cost B associated with them, that is, B is the cost of "clicking" on a suggested condition or executing an EXPAND action on the attribute $A_i \in S_R$.

If the exact sequence of actions followed by the user in navigating R_Q were known *a priori*, we could accurately determine the cost of navigation. Since this sequence is not known in advance, we estimate the navigation cost, taking into account the inherent uncertainty in the user navigation. To estimate the navigation cost, we introduce three probabilities that capture the uncertainty in user actions and are estimated in Section 5:

- **SHOWRESULT Probability $P_{SR}(R_Q)$:** This is the probability that the user examines all tuples in the result set R_Q and thus terminates the navigation. If no facet conditions are suggested, then $P_{SR}(R_Q) = 1$.
- **REFINE Probability $P(c)$:** This is the probability that the user refines the query Q by a suggested condition $c \in C_S(R_Q)$.
- **EXPAND Probability $P_E(R_Q)$:** The probability that the user does not choose a suggested condition and instead performs an EXPAND action is $P_E(R_Q) = \prod_{c \in C_S(R_Q)}(1 - P(c))$.

Since the navigation model is recursive, the expected navigation cost can be estimated by the following recursive cost formula:

$$cost(Q) = P_{SR}(R_Q) \cdot |R_Q| + \left(1 - P_{SR}(R_Q)\right) \cdot \left[\begin{array}{c} B + |C_S(R_Q)| + (1 - P_E(R_Q)) \cdot refine\left(Q, C_S(R_Q)\right) + \\ P_E(R_Q) \cdot \sum_{A_i \in S_R} P_A(A_i) \cdot \left(|C(A_i) \backslash C_S(R_Q)| + refine\left(Q, C(A_i) \backslash C_S(R_Q)\right)\right) \end{array}\right] \quad (1)$$

where

$$refine(Q, C) = \sum_{c \in C}\left(P_{norm}(c) \cdot cost(Q \wedge c)\right) \quad (2)$$

The first line of Equation 1 captures the fact that the user has two options, when presented with a set of suggested conditions. One is to execute a SHOWRESULT action with probability $P_{SR}(R_Q)$ and cost $|R_Q|$. The other is to execute a REFINE or EXPAND action with probability $1 - P_{SR}(R_Q)$. The cost entailed by this last option consists of the following parts shown in the square brackets of the cost formula:

1. A fixed cost of B of a REFINE action, that is, clicking on a facet condition.
2. The user reads the suggested conditions with cost $|C_S(R_Q)|$.
3. With probability $1 - P_E(R_Q)$ the user decides to REFINE. The cost of REFINE, shown in Equation 2, is the sum of all possible REFINE choices weighted by their probabilities. These probabilities are normalized to sum to 1, as follows:

$$P_{norm}(c) = \frac{P(c)}{\Sigma_{c \in C} P(c)}$$

4. With probability $P_E(R_Q)$, the user does not choose any of the suggested conditions and performs an EXPAND action instead (third line of Equation 1). With probability $P_A(A_i)$, the user prefers attribute A_i over all other attributes and EXPANDs it. She examines all the non-suggested conditions for A_i, $|C(A_i) \backslash C_S(R_Q)|$ in total, chooses one of them and refines query Q. The estimated cost for the last step is also given by the refine formula in Equation 2 above, where $C = C(A_i) \backslash C_S(R_Q)$.

3.3 Implications of the Cost Model

In this section, we discuss the implications of the cost model and give an intuition about the characteristics of facet conditions that are selected as the set of suggested conditions. Consider a sample result set R_Q shown in Figure 3. Also shown, are three *alternative* sets of suggested conditions (Figure 3a, 3b and 3c) selected from the set of all facet conditions $C(R_Q)$.

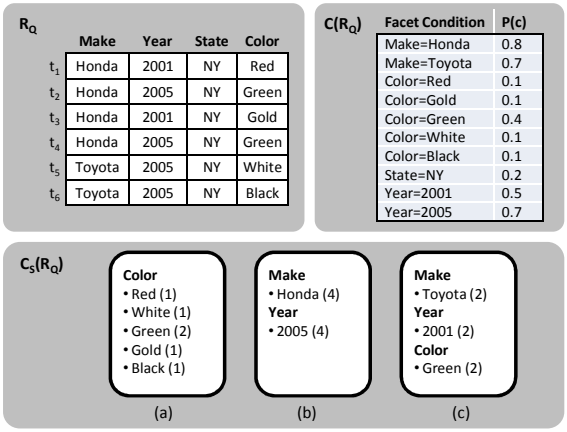


Figure 3. Result Set R_Q, All Facet Conditions $C(R_Q)$, and Three Alternative Sets of Suggested Conditions $C_S(R_Q)$

A naïve algorithm to find the optimal suggested conditions would compute the estimated cost of every possible set of suggestions and output the one with minimum cost. Which one of the alternative set of suggestions shown in Figures 3a, 3b and 3c has the lowest cost, and therefore is more likely to be selected by the navigation cost model?

The suggested conditions shown in Figure 3a are highly selective, since each one of them appears in a small number of results (low cardinality). Therefore, a large number of such conditions are required to *cover* the result set R_Q causing the navigation cost to increase as the user now has to read all the labels before proceeding to the next navigation step.

A set of suggested conditions where each condition has low selectivity (Figure 3b) also leads to a high overall expected navigation cost. Such conditions typically have a high overlap and do not effectively narrow down the result set and therefore, the user has to execute more REFINE actions to narrow down the result set. For example, refining by either $Make = Honda$ or $Year = 2005$, in Figure 3b, reduces the number of results from the initial six to four, and the resulting result set may need to be refined further before reaching the desired result(s). Conditions with low selectivity can potentially lead to redundant navigation steps. For example, refining by $State = NY$ does not narrow down the result but still adds to the navigation cost.

Based on the above discussion, we observe that the facet conditions selected by the cost model as suggested ones should neither have high nor low selectivity. The suggested conditions in Figure 3c are facet conditions with such desired characteristics. The conditions $Make = Toyota$, $Year = 2001$ and $Color = Green$ are moderately selective and thus have minimum overlap and do not require a large number of conditions to cover R_Q.

Another factor that increases the navigation cost is the EXPAND action, since the user can potentially see a large number of conditions, thereby increasing the navigation cost. The expected cost of EXPAND is multiplied by $\prod_{c \in C_S(R_Q)}(1 - P(c))$, which is minimized when all the conditions in $C_S(R_Q)$ have a high $P(c)$.

4. COMPLEXITY RESULTS

In this section, we study the complexity of the *Facet Selection* problem.

Problem 1 (Facet Selection): Given a query Q and a result set R_Q, find the set $C_S(R_Q)$ of suggested facet conditions that minimizes the expected navigation cost for the NAVIGATE navigation model described in Section 3.1.

We prove that this problem is NP-Hard, by showing that a simplified version of the problem is also NP-Hard. The *Simplified Facet Selection (SFS)* problem considers a simpler navigation model, NAVIGATE-SINGLE, defined next.

NAVIGATE-SINGLE: In NAVIGATE-SINGLE, the system performs a single REFINE action, where the user randomly selects one of the suggested conditions, and then performs a SHOWRESULT action. The cost of NAVIGATE-SINGLE navigation is the cost to examine all suggested conditions displayed ($|C_S(R_Q)|$) plus the cost $|R_{Q \wedge c}|$ of performing the SHOWRESULT action for the randomly-selected suggested condition c.

Suppose that the dominant cost of our cost model is that of examining a suggested condition. That is, suppose the cost to examine a suggested condition is 1 and the cost of SHOWRESULT is 0. Also suppose that all attributes of R_Q are Boolean (0, 1) and that the suggested conditions in $C_S(R_Q)$ are always positive, that is, $A_i = 1$. Recall that facet conditions only specify a single attribute.

Theorem 1: *The SFS problem is NP-Hard.* □

Proof Intuition: In this simplified problem we minimize the cost by computing the minimum number of facet conditions that partition the result set R_Q such that every tuple in R_Q satisfies at least one of these conditions.

Proof: SFS is clearly in NP. To prove the NP-Hardness we reduce the HITTING-SET problem to the SFS problem. First, we define the HITTING-SET problem.

An instance of the HITTING-SET problem consists of:

- a hypergraph $H = (X, E)$, where X is a finite set of vertices and $E = \{E_1, \dots, E_n\}$ is a set of hyperedges, that is, subsets of X, and
- a positive integer $l \leq |X|$.

The problem is to determine whether there is a hitting set $H \subseteq X$ of size l such that $\forall i \in \{1, \dots n\}: H \cap E_i \neq \emptyset$.

We reduce HITTING-SET to SFS as follows. A node u_i in X becomes a facet condition $A_i = 1$. A hyperedge $E_i \in E$ becomes a tuple t_i in the result set R_Q. E_i connects the vertices corresponding to the attributes that have value 1 for the result t_i. The solution of HITTING-SET translates naturally to a solution to NAVIGATE-SINGLE and vice versa. □

5. ESTIMATING PROBABILITIES

In order to compute the expected navigation cost using the cost formula in Equation 1 (Section 3.2), the probabilities $P_{SR}(R_Q)$, $P_A(A_i)$ and $P(c)$ must be estimated.

Estimating $P_{SR}(R_Q)$, the probability that the user executes SHOWRESULT on a given result set R_Q. We use the information theoretic measure of *Entropy* to estimate P_{SR}. The rationale behind this decision is that the user would choose to further refine the query Q and narrow down the result set R_Q if the tuples in R_Q are widely distributed among all possible facet conditions $C(R_Q)$. The entropy of a result set R_Q distributed amongst the facet conditions in $C(R_Q)$ is given by:

$$Entropy\left(R_Q, C(R_Q)\right) = -\sum_{c \in C(R_Q)} \left(|R_{Q \wedge c}|/N\right) \ln\left(|R_{Q \wedge c}|/N\right)$$

where $N = \sum_{c \in C(R_Q)} |R_{Q \wedge c}|$ is the sum of the number of tuples over all facet conditions.

Since the value of entropy can be greater than 1, we normalize it with the maximum value of entropy for a given result set R_Q distributed over $|C(R_Q)|$ facet conditions. Entropy is maximal when N tuples are distributed equally amongst $|C(R_Q)|$ facet conditions, that is, each facet condition is satisfied by $N/|C(R_Q)|$ tuples. The total entropy of such a system is:

$$Entropy_{MAX}\left(R_Q, C(R_Q)\right) = -\sum_{c \in C(R_Q)} \frac{\frac{N}{|C(R_Q)|}}{N} \ln \frac{\frac{N}{|C(R_Q)|}}{N} = \ln|C(R_Q)|$$

Hence,

$$P_{SR}(R_Q) = \frac{-\sum_{c \in C(R_Q)} \frac{|R_{Q \wedge c}|}{N} \ln \frac{|R_{Q \wedge c}|}{N}}{\ln|C(R_Q)|}$$

The above formula for P_{SR} does not distinguish between the sizes of result sets. Intuitively, a user is very likely to perform a SHOWRESULT action if $|R_Q|$ is very small, and a REFINE action if $|R_Q|$ is very large. For these cases, we use upper and lower thresholds as follows:

$$P_{SR}(R_Q) = \begin{cases} 1 & ,if\ |R_Q| \leq 10 \\ 0 & ,if\ |R_Q| \geq 50 \\ \dfrac{-\sum_{c \in C(R_Q)} \frac{|R_{Q \wedge c}|}{N} \ln \frac{|R_{Q \wedge c}|}{N}}{\ln |C(R_Q)|} & ,otherwise \end{cases}$$

Estimating $P_A(A_i)$, the probability that the user chooses attribute A_i over all other attributes. $P_A(A_i)$ is a subjective measure of user's preference and is estimated in multiple ways, such as eliciting them from users, as we did for this work, or learning from navigation patterns. We provide more details in Section 7.1.

Estimating $P(c)$, the probability that the user REFINEs with a facet condition c. To estimate $P(c)$, we fist use the attribute level probabilities $P_A(A_i)$ estimated above. Then, the individual values within an attribute are assigned REFINE probabilities proportionally to their frequency in the database.

6. ALGORITHMS

An optimal set $C_S(R_Q)$ of suggested conditions for a given R_Q can be naively calculated by considering at each navigation step all combinations of candidate facet conditions from $C(R_Q)$ and recursively computing the cost formula in Equation 1. However, this naïve approach is exponential on the size of $C(R_Q)$ since all combinations of conditions are considered at each navigation step.

We present two heuristics to efficiently compute the best set of suggested conditions. The first, ApproximateSetCover (Section 6.1), is inspired by an approximation algorithm for the weighted set cover problem [8], and attempts to find a relatively small set of suggestions that have a high probability of being recognized by users (high $P(c)$). A drawback of this heuristic is that it does not closely resemble the cost formula in Equation 1, since it cannot incorporate the cost parameter B and the uncertainty in the cost expression. The second heuristic, UniformSuggestions (Section 6.2), which follows Equation 1 more closely, greedily selects each facet condition c assuming that all future suggestions have identical properties as c. In Section 7, we present an evaluation of the two heuristics and show that both of these heuristics generate better faceted conditions than the state of the art.

6.1 ApproximateSetCover Heuristic

For a given result set R_Q and all its facet conditions $C(R_Q)$, the objective is to compute the set of suggested conditions $C_S(R_Q)$ such that the expected navigation cost, based on our cost model for user navigation, is minimal and the set $C_S(R_Q)$ *covers* R_Q, that is, $\bigcup_{c \in C_S(R_Q)} R_{Q \wedge c} = R_Q$, where each facet condition c covers $|R_{Q \wedge c}|$ results in R_Q.

This problem closely resembles the well-known NP-hard weighted set cover problem – given a set system (U, S), such that $\bigcup_{s \in S} s = U$, and weights $w: S \to \mathbb{R}_+$, find a subfamily $\mathcal{F} \subseteq S$ such that $\bigcup_{s \in \mathcal{F}} s = U$ and $\sum_{s \in \mathcal{F}} w(s)$ is minimal. The approximation algorithm for weighted set cover [8], adds at every step the set s that maximizes the number of newly covered elements divided by the weight $w(s)$.

In order to apply the approximation algorithm for weighted set cover to our problem, we need to define the weight function $w: C(R_Q) \to \mathbb{R}_+$. By observing the cost formula in Equation 1, each facet condition in the suggested set $C_S(R_Q)$ should have a high probability $P(c)$ of being selected for REFINEment. Otherwise, the probability that the user does not select a suggested condition and chooses to execute a EXPAND action would be high, resulting in a high overall cost. To achieve this objective, we set the weight function to be:

$$w: c \in C(R_Q) \to 1/P(c)$$

Note that the overlap among conditions and number of elements covered by a selected condition do not need to be part of w, since they are considered directly in the approximation algorithm.

Algorithm: ApproximateSetCover(Q, R_Q)
Input: A query Q, a result set R_Q
Output: The suggested conditions $C_S(R_Q) \subseteq C(R_Q)$

```
1  C_S(R_Q) ← ∅
2  V ← ∅                 // V are results covered so far
3  while V ≠ R_Q         // while not all results covered
4        c ← argmax_{c∈C(R_Q)}(P(c) · |R_{Q∧c} \ V|)
5        C_S(R_Q) ← C_S(R_Q) ∪ {c}
6        V ← V ∪ R_{Q∧c}
7        Q ← Q ∧ c
8  return C_S(R_Q)
```

Figure 4. ApproximateSetCover Heuristic

Figure 4 presents the ApproximateSetCover heuristic, which is an adaptation of the weighted set cover approximation algorithm [8] using the above defined weight function, and has a running time of $O(|C(R_Q)| \cdot |R_Q|)$ and an approximation ratio of $O(\log(|C(R_Q)|))$. Note that this approximation ratio assumes that the quantity we want to minimize is the sum of the weights $(1/P(c))$ of the selected conditions. However, the real objective of ApproximateSetCover is to minimize the navigation cost, which is much harder to bound, given that ApproximateSetCover does not capture all the details of Equation 1. Also note that this approximation ratio can be large if the number of conditions in $C(R_Q)$ is large. However, the number of facet conditions is generally small and this algorithm performs reasonably well in practice, as demonstrated by the experiments in Section 7.

Example Figure 3b shows the result of the ApproximateSetCover heuristic on the result set R_Q in Figure 3. The algorithm requires two iterations of the while loop (lines 3-7) before terminating with the set of suggested conditions in Figure 3b. In the first iteration, the algorithm selects Make = Honda, since this facet condition covers 4 results and has the maximum value of $P(c) \cdot |R_{Q \wedge c}| = 3.2$ amongst all the conditions in $C(R_Q)$ and V is empty. In the next iteration, two results (t_1 & t_3) remain uncovered and are covered by facet condition Year = 2005. □

6.2 UniformSuggestions Heuristic

In this heuristic we follow the cost formula in Equation 1 more closely, which leads to a more robust heuristic as we show in Section 7. As mentioned earlier, computing the optimal suggested conditions involves recursively evaluating Equation 1 for each combination of facet conditions in $C(R_Q)$. This translates to a very large (in both height and width) recursion tree. UniformSuggestions replaces this recursion tree with a set of very

small recursion trees, one for each condition in $C(R_Q)$. For that, we evaluate the expected cost of each facet condition independently, assuming that all future suggested conditions will have identical properties, and then select the facet conditions with minimal expected cost, until all results in R_Q are covered.

In particular, the *uniform-condition* heuristic assumption states that for a given condition $c \in C(R_Q)$, evaluate the navigation cost using Equation 1, while assuming that every other condition in $C(R_Q)$ has the same *characteristics* as c. The characteristics of c are *(a)* its probability $P(c)$, and *(b)* the ratio $r(c) = |R_{Q \wedge c}|/|R_Q|$ of the uncovered results that c covers. This heuristic assumption reduces the search space of suggestions to $|C(R_Q)|$ as each condition is now evaluated independently. It also allows us to simplify the cost formula in Equation 1.

If each suggested condition in $C_S(R_Q)$ covers a ratio r of the results in R_Q, we need a total of $n = 1/r$ conditions to cover all the results in R_Q. Also, REFINEment by c narrows down R_Q to an estimated $|R_Q|/n$ number of results. On the other hand, if the user does not select a suggested condition and instead EXPANDs an attribute A_i, she views an additional $|C(A_i) \backslash C_S(R_Q)| \approx |C(A_i)|$ facet conditions. Also, in the absence of any prior knowledge about the selectivity of facet conditions in $C(A_i)$, we assume that each $c' \in C(A_i)$ narrows down R_Q to an estimated $|R_Q|/|C(A_i)|$. Thus, we can simplify the recursion in Equation 1 as follows:

$$cost(c, |R_Q|) = P_{SR}(R_Q) \cdot |R_Q| + \left(1 - P_{SR}(R_Q)\right) \cdot \left[\begin{array}{l} B + n + \left(1 - P_E(c)\right) \cdot \sum_{i=1}^{n} \left(P_{norm}(c) \cdot cost(c, |R_Q|/n)\right) + \\ P_E(c) \cdot \sum_{A_i \in S_R} P_A(A_i) \cdot \left(|C(A_i)| + refine(C(A_i), |R_Q|)\right) \end{array} \right] \quad (3)$$

where $P_E(c) = \left(1 - P(c)\right)^n$ and

$$refine(C(A_i), |R_Q|) = \sum_{c' \in C(A_i)} \left(P_{norm}(c') \cdot cost(c', |R_Q|/|C(A_i)|)\right) \quad (4)$$

Observe that instead of Q, the cost function in Equation 3 above uses c and $|R_Q|$ as arguments for this heuristic, since a cost is computed for each c, and only the number of results $|R_Q|$ is important. The parameter Q in the original cost formula (Equation 1) captured the query progression with REFINE actions, which is not required in this heuristic, since only the result pruning at each step is important and not the query itself.

In Equation 3 above, $P_{norm}(c)$ is the normalized probability of following one condition of *type* c. Since all n suggested conditions have the same $P(c)$, then $P_{norm}(c) = 1/n$. Therefore the cost component in Equation 3 for navigating all n suggested conditions can be rewritten as:

$$\sum_{i=1}^{n} P_{norm}(c) \cdot cost(c, |R_Q|/n) = cost(c, |R_Q|/n)$$

By a similar argument, and since every facet condition c' in Equation 4 has the same characteristics as c in Equation 3, we can simplify Equation 4 as follows, where A_c is the attribute of facet condition c:

$$refine(C(A_i), |R_Q|) = |C(A_c)| + cost(c, |R_Q|/|C(A_c)|)$$

By the *uniform-condition* heuristic assumption, all the *attributes* of suggestions have the same characteristics. Therefore, we can reasonably assume that each attribute A_i has $P_A(A_i) = 1/|S_R| = P_A(A_c)$. Hence, the following simplification is possible:

$$\sum_{A_i \in S_R} P_A(A_i) \cdot refine(C(A_i), |R_Q|) = |C(A_c)| + cost(c, |R_Q|/|C(A_c)|)$$

Based on the above, the cost formula in Equation 3 can now be re-written as:

$$cost(c, |R_Q|) = \begin{cases} |R_Q| & , |R_Q| < T \\ P_{SR}(R_Q) \cdot |R_Q| + \left(1 - P_{SR}(R_Q)\right) \cdot \left[\begin{array}{l} B + n + \left(1 - P_E(c)\right) \cdot cost(c, |R_Q|/n) + \\ P_E(c) \cdot \left(|C(A_c)| + cost(c, |R_Q|/|C(A_c)|)\right) \end{array} \right] & , |R_Q| > T \end{cases} \quad (5)$$

The recursion terminates when the size of the result $|R_Q|$ drops below a threshold T. Since a navigation should be able to narrow down the result to a single tuple, we set T to 1.

Algorithm: UniformSuggestions(Q, R_Q)
Input: A query Q, a result set R_Q
Output: $C_S(R_Q) \subseteq C(R_Q)$, the suggested conditions.

```
1  Q' ← Q; C_S(R_Q) ← ∅; Y ← R_Q            // Y: uncovered results
2  P_SR ← P_SR(R_Q, C(R_Q))
3  while Y ≠ ∅ do
4    foreach c ∈ C(R_Q)
5      n ← |Y|/|Y ∩ R_{Q'∧c}|
6      P_SR ← P_SR(R_Q)
7      u ← |Y|
8      compute cost(c, u) using Equation 5
9    endFor
10   Let cmin be the suggestion with min estCost(c, |Y|)
11   C_S(R_Q) ← C_S(R_Q) ∪ cmin
12   Q' ← Q ∧ cmin
13   Y ← Y \ R_{Q'∧cmin}
14   C(R_Q) ← C(R_Q) \ {cmin}
15 endWhile
16 return C_S(R_Q)
```

Figure 5. UniformSuggestions Heuristic

The algorithm, based on the *uniform-condition* heuristic assumption is presented in Figure 5. The algorithm computes the estimated $cost$ of each facet condition using the simplified cost formula in Equation 5 (lines 4-9), and selects the condition with the minimum $cost$ ($cmin$) to be added to the set of selected conditions (lines 10-11). Next, we remove from the set V of uncovered results the results covered by $cmin$. The algorithm terminates when all the results in R_Q are covered.

The result of applying the UniformSuggestions heuristic algorithm to the result set R_Q in Figure 3 is shown in Figure 3c. Recall from the discussion in Section 3.3 that the cost model selects conditions with *moderate* selectivity and high $P(c)$. Under our heuristic assumption, a facet condition c is evaluated under the assumption that all conditions in $C(R_Q)$ have the same characteristics as c. Therefore, a condition with *moderate* selectivity and a high $P(c)$ has a lower cost when evaluated using the simplified cost formula in Equation 5 and these are just the conditions selected by the algorithm.

7. EXPERIMENTAL EVALUATION

In this section, we present a thorough evaluation of the algorithms and heuristics described in Section 6. We show that FACeTOR achieves a significant decrease in navigation cost compared to current approaches. The experiments are based on a large-scale simulation of user navigations presented in Section 7.2. The metric used is the average navigation cost as defined by the cost formula in Equation 1. In Section 7.1, we describe the experimental setup, including the choice of datasets. Section 7.3 measures the time requirements of our heuristics and shows that they can be used for real-time interaction.

7.1 Experimental Setup

The primary goal of these experiments is to evaluate the effectiveness of the system in decreasing the user navigation cost for a set of query results. To this end, we compare the two heuristics presented in Section 6 to each other and to the current state of the art algorithm, which is the single-facet-based-search [22], henceforth called INDG.

The experiments reported in this section were conducted on a Dell Optiplex machine with 3GHz CPU and 3GB of RAM. We use MySQL as our database and Java for the algorithms.

7.1.1 Datasets

We evaluate FACeTOR on two datasets, *UsedCars* and IMDB. We assume that the numeric attributes have been appropriately discretized. The *UsedCars* database was downloaded from Yahoo! Autos site and contains 15,191 car tuples with 41 attributes/facets, of which 7 are categorical, 3 numerical, and the rest Boolean.

From the IMDB dataset, we extracted a total of 37,324 movies. For our experiments we only leveraged the movie, actors, directors, ratings and genre data. Note that actors, directors and genres are set-valued attributes, that is, each movie can have multiple actors and/or directors. These set attributes can be problematic in a faceted search and navigation settings as these techniques are biased towards results that have a large number of facet conditions [9]. We use the *Binarization* technique presented in [9] in dealing with set-valued attributes.

7.1.2 Data Pre-processing

From the initial dataset relation R, we extract all the candidate facet conditions $C(R)$ and store them in a relation with schema $< attribute, value, count >$, where $count$ stores the number of tuples in R that satisfy the facet condition c with the given $attribute/value$ combination. This value is used to compute the estimate of $P(c)$ of each facet condition $c \in C(R)$.

Computation of $P_A(A_i)$ This is the probability that the user knows or likes attribute A_i. We estimated this probability using a survey of 10 users (students and faculty in our institutions) who rated each attribute A_i in the dataset on a scale from 0 to 1. These values are taken to be the user preference $P_A(A_i)$ for attribute A_i.

Computation of $P(c)$ As defined in Section 3, $P(c)$ is the probability that the user executes a REFINE action on suggested condition c. A user would choose to REFINE by c, if she knows or likes the attribute of c and is also familiar with the value of the attribute in c. Therefore, we use a two-pronged approach to compute $P(c)$. To estimate the popularity of a value of a facet condition, we computed the frequency $freq(A_i.v_i)$ of each value for each attribute in R. Then, we multiply each frequency with the attribute preference to obtain the attribute/value preferences $P(c: A_i = v_i) = freq(A_i.v_i) \cdot P_A(A_i)$, which we then normalize by dividing by the maximum frequency for each attribute.

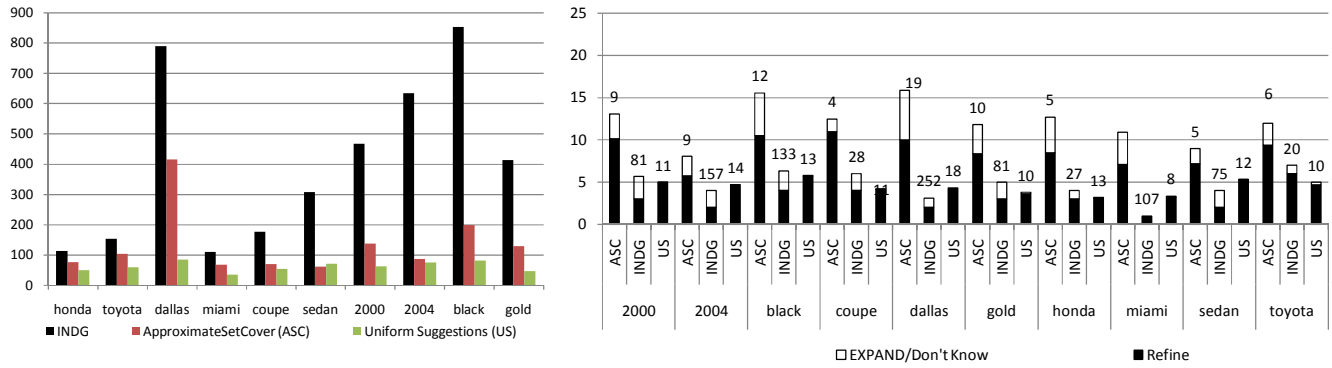
Table 2. Query Workload

	Query	#Results	#of Facet Conditions
***UsedCars* DataSet**			
1	honda	789	234
2	toyota	1470	366
3	dallas	2932	990
4	miami	211	230
5	coupe	599	334
6	sedan	1693	524
7	2000	896	641
8	2004	3711	1124
9	black	2391	972
10	gold	709	508
***IMDB* DataSet**			
11	baldwin	112	1545
12	oscar	189	2141
13	love	415	2989
14	American	111	1096
15	history	272	2716
16	white	284	3058
17	black	221	2327
18	time	145	907
19	john *and* 2007	391	4545
20	action *and* 2007	272	2601

7.1.3 Experimental Methodology

For each dataset, *IMDB* and *UsedCars*, we select a number of keyword queries (see Table 2) whose results form the initial result set R, and a random result tuple as the target for navigation for each query. Next, we measure the number of navigation actions (REFINE/EXPAND actions, facet conditions displayed and results viewed) incurred before reaching the target tuple as the navigation cost for the query. In our system, the target tuple can be reached by multiple navigations. For example, tuple t_4 in the result set of Figure 3 can be reached by REFINEing by any one of the two conditions in Figure 3b.

Since, the user's navigation cannot be known in advance, we consider an evaluation approach that considers both these navigation paths. To account for uncertainty in user navigation, we use a *guided randomized simulation* of user navigation. In this simulation, we randomly select one of the facet conditions $c \in C_S(R_Q)$ for navigation. The probability that the agent selects a condition c is proportional to $P(c)$, the probability that the user would know or likes the facet condition c. The simulation is guided in the sense that it only follows the paths that lead to the target result. For example, if the agent encounters the two suggestions in Figure 3b and the target is tuple t_3, the simulation would choose either $Make = Honda$ or EXPAND ("Don't Know" for INDG). The probability of choosing EXPAND is $\prod_{c \in C_S(R_Q)}(1 - P(c))$, where $C_S(R_Q)$ are the suggested conditions. If an EXPAND action is chosen, the agent selects an attribute $A_i \in S_R$ with probability $P_A(A_i)$, and selects a condition $c' \in R_Q(A_i)$. In the *UsedCars* dataset, there is only one condition that can be chosen, since the facet conditions on an attribute partitions the result-set completely. However, in the IMDB dataset, there can be multiple conditions that can be followed. In this case the agent chooses a condition with probability proportional to $P(c')$.



(a) (b)

Figure 6. For the *UsedCars* Dataset: (a) Average Navigation Cost, and (b) Average Number of REFINE and EXPAND Actions, and Average Number of Suggested Conditions per Navigation Step (numbers on top of the bars), for $B = 1$

We execute the navigation for each query 1000 times using this simulation technique and average the cost over the individual navigations. We also report the average number of times each navigation action is executed during the simulation.

The navigation cost is sensitive to the constant B according to the cost function in Equation 1. Varying these constants changes the set $C_S(R_Q)$ for UniformSuggestions, but not for INDG or ApproximateSetCover, since these algorithms do not consider B. Intuitively, B denotes the patience of the user towards suggestions generated by the system. If the user sees a small number of conditions she would have to execute more REFINE actions to reach the result. Also, the chance that she would have to execute the EXPAND action also increases. Thus by setting B to a large value the user should typically see more suggestions per REFINE and vice versa. We experiment with different values of B and observe the effect on the overall navigation cost for the *UsedCars* query workload in Table 2. We also compare the number of suggested conditions generated (on average) and the number of REFINE actions.

We compare our approach with the current state of the art INDG algorithm [22]. This algorithm constructs a decision tree that partitions the result set R_Q by a facet (attribute) at each level. The aim is to minimize the average depth of the decision tree in reaching the results. The user is presented with all the facet conditions on the attribute that forms the root of the decision tree. The system proposes to pre-compute the decision tree for a predefined set of queries. The navigation in these cases will be essentially fixed for a given query and does not depend on user refinements. To ensure fairness, we modify the approach to re-compute the decision tree at each REFINE step, even though this is computationally expensive.

7.2 Experiments with Navigation Cost

The average navigation costs for the INDG, ApproximateSetCover and UniformSuggestions algorithms for the *UsedCars* queries in Table 2 are shown in Figure 6a. As seen in the graph, the navigation cost incurred by following our approach leads to significant savings in cost as compared to the existing approach. The INDG algorithm ignores the cost of inspecting suggested conditions and thus produces on average a large number of suggestions at each navigation step.

Figure 6b shows some of the individual components of the total cost for Figure 6a, that is, the average number of REFINE actions, of EXPAND actions for ApproximateSetCover and UniformSuggestions and of "Don't Know" actions for the INDG algorithm. Also shown on top of the bars in Figure 6b are the average numbers of suggestions per navigation step. As expected, the INDG algorithm has very few REFINE and "Don't Know" actions, but reveals a large number of facet conditions in each navigation step, resulting in high total cost.

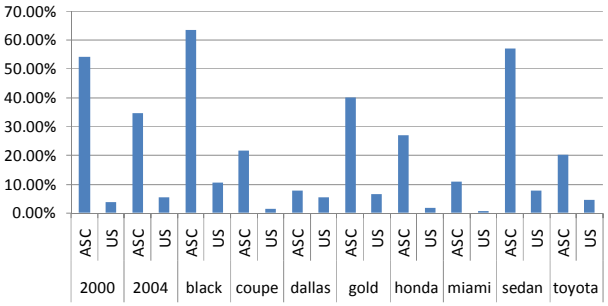


Figure 7. Average Overlap per Navigation Step for the *UsedCars* Dataset, for $B = 1$

The average cost incurred by UniformSuggestions algorithm is less compared to ApproximateSetCover. ApproximateSetCover has a higher number of REFINE and EXPAND actions as compared to UniformSuggestions, even though the average number of suggestions at each navigation step is comparable. In each iteration, the greedy ApproximateSetCover algorithm selects a small set of facet conditions with a high value of $P(c)$ that also cover a large number of uncovered results. These suggested conditions have thus a low selectivity and therefore tend to have a high degree of overlap among the suggested conditions, as shown in Figure 7, thereby reducing the effectiveness of REFINE actions. Thus, the user has to perform many REFINE actions in order to reach the target result. Also, since the number of suggestions is small, the chance that the user executes an EXPAND action is also significant. These EXPAND actions are costly, since the user now has to read a (potentially) large number of facet conditions.

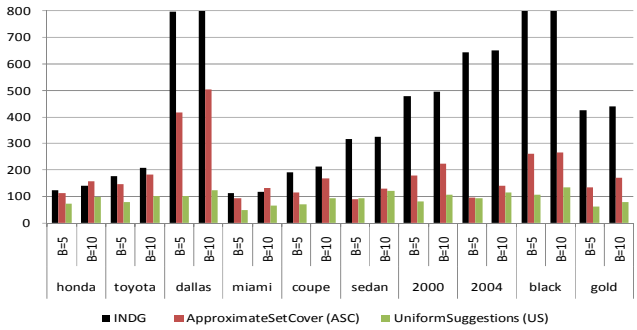


Figure 8. Average Navigation Cost for $B = 5$ and $B = 10$ (*UsedCars* Dataset)

Figure 8 shows the effect of increasing B, the cost of executing a REFINE. As expected, the average overall cost increases. The UniformSuggestions heuristic adapts to a changing value of B, whereas the ApproximateSetCover heuristic and INDG do not. Therefore the cost of UniformSuggestions increases at a slower rate than the other two algorithms. This is primarily because, for a higher B, UniformSuggestions generates more suggestions per REFINE/EXPAND. While INDG is insensitive to B, it reveals a large number of labels, has very few REFINE/EXPAND actions, and therefore the overall cost increases at a slow rate.

The results of *IMDB* workload queries in Table 2 are shown in Figure 9. The INDG algorithm assumes that the facet classification on an attribute partitions the result set. This is not true in our *IMDB* dataset where a single movie can be classified into multiple actors. Therefore, we cannot evaluate the INDG algorithm for this dataset. As in the *UsedCars* workload, the UniformSuggestions heuristic outperforms ApproximateSetCover. Also, the observations for the number of EXPAND and REFINE actions and the number of suggested conditions generated are also similar to those for the *UsedCars* dataset. However, the navigation cost with the UniformSuggestions algorithm is much lower than ApproximateSetCover. A *movie* in the IMDB dataset can be classified into a large number of facet conditions. For example, each movie can have multiple actors or directors or genres. Therefore executing an EXPAND action reveals a very large number of facet conditions (the number on top of bars in Figure 9b), thereby significantly increasing the navigation cost.

7.3 Execution Time Evaluation

This experiment aims to show that UniformSuggestions is fast enough to be used in real-time. The average execution time of UniformSuggestions per REFINE action for the queries in Table 2 (*UsedCars* dataset) is shown in Figure 10. The execution time for this heuristic depends primarily on the number of facet conditions in the result set R_Q. As the number of facet conditions decreases, as is the case towards the end of navigation, the performance of UniformSuggestions improves dramatically. In the interest of space, we omit reporting these values, as well as the results for ApproximateSetCover which, given its simplicity, is much faster.

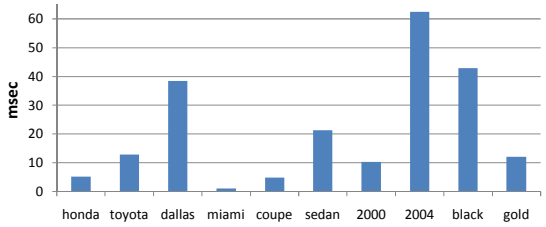


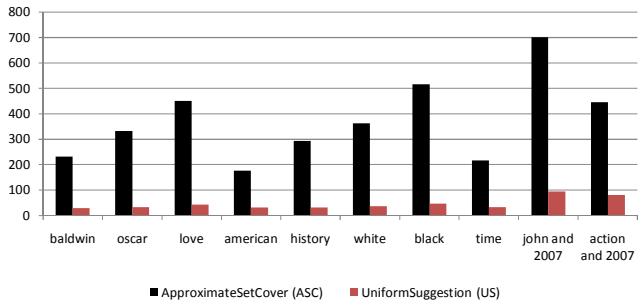
Figure 10. Average Execution Time of UniformSuggestions Heuristic (UsedCars dataset & $B = 1$)

8. USER EVALUATION

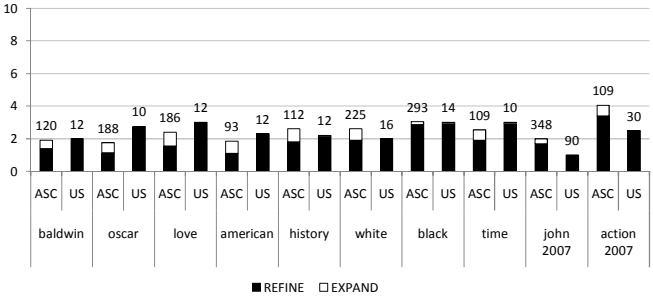
In this section, we present the results of a large scale user study we conducted to compare the user experience with FACeTOR and other state of the art interfaces. We measure the following: *(a)* the *actual time* it took users to navigate to designated target tuples using different interfaces, *(b)* how realistic is our cost model, by studying the relationship of the actual time (actual cost) with the *estimated cost* as computed by the cost formula in Equation 1, and *(c)* the *users perception* of the faceted interfaces through a questionnaire. By comparing the actual navigation time to the users' perception, we study if lower actual time corresponds to more intuitive (cognitively easier) interfaces or if the users' familiarity with some interfaces skews their opinion.

The results show that FACeTOR achieves significant savings in the effort required to navigate the query results, both in terms of actual time and estimated navigation cost. Through the questionnaire, a large majority of users also confirmed that the FACeTOR interface is intuitive and rated it higher in terms of quality of suggestions as compared to other interfaces.

Setup We constructed 8 randomly created result sets of 1000 tuples from the *UsedCars* dataset and for each one we created a task that involves locating a set of 3 or fewer *target tuples* (cars), which satisfy a set of attribute/value conditions. For each one of



(a)



(b)

Figure 9. For the IMDB Dataset: (a) Average Navigation Cost, and (b) Average Number of REFINE and EXPAND Actions, and Average Number of Suggested Conditions per Navigation Step (numbers on top of the bars), for $B = 1$

the 8 result sets, we showed the requested conditions to the users and asked them to locate the target tuples using three interfaces:

- *FACeTOR*,
- *Amazon-Style*, which suggests at most 5 facet conditions with the highest cardinality for each attribute, and
- One-attribute-at-a-time *INDG* [22], where an attribute is selected at each step and all its conditions are displayed.

We deployed our system as an Amazon Mechanical Turk [3] task and collected a total of 37 valid responses.

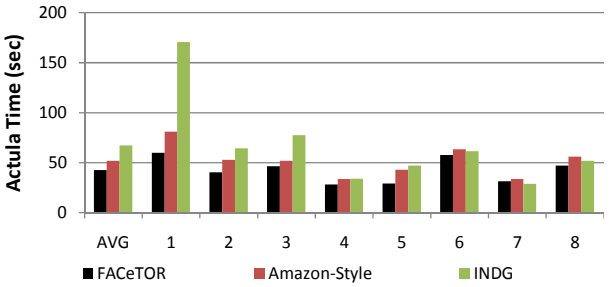


Figure 11. Actual User Navigation Time for 8 Result Sets

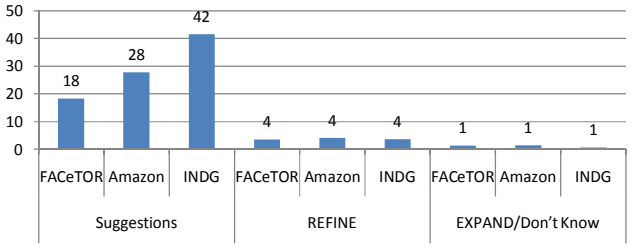


Figure 12. Average # of Suggestions and Actions

Actual Time Figure 11 shows the actual time taken by users to navigate each of the eight result sets using the three interfaces. Also shown is the average actual time for each interface. As shown, FACeTOR speeds up the navigation by 18% and 37% over Amazon-Style and INDG respectively, even for relatively small result sets of 1000 tuples and short navigations consisting of only 4 REFINE actions (Figure 12). This is primarily because users spend less time in reading suggested conditions and deciding which one to follow next, as evidenced by Figure 12. FACeTOR shows 36% fewer suggestions than Amazon-style and 57% fewer suggestions than INDG, while it requires the same number of REFINE and EXPAND actions (on average) to reach the target tuples. This is an indication of *high quality* suggestions provided by FACeTOR.

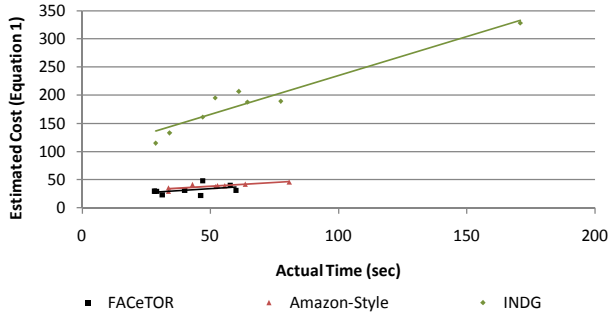


Figure 13. Actual Time vs. Estimated Navigation Cost

Estimated Cost Figure 13 displays the data points of actual time vs. estimated cost, as computed by Equation 1, for the eight result sets for the three interfaces. Based on these data points, Figure 13 also shows the trend line between actual time and estimated navigation cost for each interface. We observe that the actual time is linearly proportional to the estimated navigation cost for all three interfaces, which shows that our cost model is realistic.

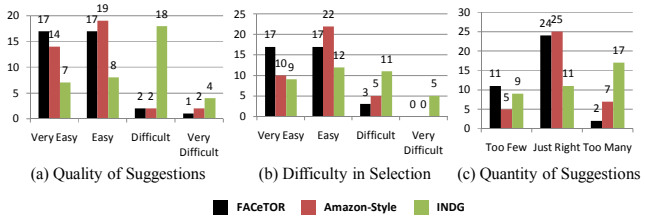


Figure 14. Users Perception (Questionnaire)

Users Perception The study also included a questionnaire where we elicited the users' opinion on various aspects of the three interfaces, including the ease of use, size and intuitiveness of suggested conditions and their preferred choice of interface. The results of this survey are shown in Figure 14. 92% of users said that they thought the suggestions presented by FACeTOR at each step made the task of locating the target tuples easier (Figure 14a), compared to 89% for Amazon-style and 40% for INDG. A large majority of users (92%) also said that the suggestions provided by FACeTOR had a low "cognitive dissonance" (Figure 14b) in the sense that it was very easy (45%) or easy (46%) to decide which suggestion to follow next. The corresponding cumulative percentages for Amazon and INDG were 81% and 54% respectively.

We also asked the users if the number of suggestions provided by the interfaces were adequate (Figure 14c). A significant percentage (30%) said that FACeTOR provided too few suggestions at each navigation step, indicating that users prefer more choices even if it means an increase in absolute navigation cost. Since FACeTOR is customizable to user's navigation patterns, this could easily be remedied by increasing the value of the constant *B* as discussed in Section 7.1.

9. RELATED WORK

Ranking Ranking could be applied in conjunction with a faceted interface. Chaudhuri et al. [7] use the unspecified attributes and apply Probabilistic Information Retrieval principles to rank the results of a database selection query. Various ranking techniques have also been proposed for keyword search on structured databases [2,17] based on the size and relevance of the results.

Faceted Search on Structured Data Faceted search is employed by major e-Commerce websites (Amazon, eBay) that typically display all the facet conditions applicable to the current set of query results. If too many values are available for a facet, then the most popular are displayed, and a "more" button reveals the rest. In contrast, our approach displays only a subset of applicable facet conditions chosen to minimize the overall navigation cost. English et al. [13] was one of the first to introduce faceted search and discusses facets from a user interface perspective.

Our work is closest to the works of Chakrabarti et al. [6] and Roy et al. [22], which also use a navigation cost based approach for faceted navigation. In particular, FACeTOR adopts ideas from both works and addresses their key shortcomings. In both these

works, the navigation algorithm selects one attribute (or possibly multiple attributes [22]) and displays *all* the values of these attributes to the user. Alternatively, a text box could be displayed for each selected attribute [22], but we believe that this is impractical, given that the user would already have specified all known values in the original query. Our approach differs from these works, because at each navigation step, we display a mix of facet conditions from several attributes, that is, our algorithm operates at the attribute value level and not the attribute level. Moreover, facet conditions suggested by FACeTOR are selected so that the overlap of the sets of results they cover is minimized. The user does not have to choose between facet conditions that could lead to the same set of results.

Ben-Yitzhak et al. [4] focuses on providing additional qualitative information for each suggested facet to help users better choose a condition. For instance, they may show the average price for each author facet condition. Their work is complementary to ours, since we could use their algorithms to display additional information for each suggested condition.

Keyword-Based Faceted Search and Query Refinement The GrowBag project [12] and Sarkas et. al [23] suggest additional search terms based on the co-occurrence patterns of these terms in the query result. The GrowBag algorithm[12] computes higher order *co-occurrences* of terms in the document collection and suggests terms appearing in the *neighborhood* of each search term as refinement suggestions whereas [23] suggests terms that co-occur with search terms and narrow down the result-set to *interesting* subsets using the *surprise* metric. Dakka et al. [10,11] propose using external resources like Wikipedia to find the best keywords to suggest to the user. [10] uses a greedy algorithm to select the set of conditions that fit in the screen and cover the maximum number of results. However, in contrast to FACeTOR, they do not use a navigation cost model to minimize the expected navigation cost. Our work is also related to query refinement systems [20,25]. [25] recommends new terms for refinement such that the *recall* of the resulting query is maximized, whereas [20] uses *relevance judgment feedback* on the results to refine the query. Our approach also suggests facet conditions to refine the query, but we use the navigation cost as metric.

Our navigation model is similar to BioNav [18], which uses the ontological annotations of PubMed publications to create a navigation tree. A key difference is that in BioNav, there is a given concept hierarchy [19], which prunes the search space. In contrast, there is not such tree in FACeTOR, which makes the selection of a set of faceted conditions harder.

OLAP A faceted interface can be viewed as an OLAP-style cube over the results. Wu et al.[26] generate hierarchical partitions over the query results based on a cost model for user navigation and display this hierarchy to the users. The interestingness of group-by aggregations is used to rank candidate aggregations to display.

10. CONCLUSIONS

Faceted navigation is widely used to avoid *information-overload* experienced by a user navigating query results. However, the number of facet conditions encountered during faceted navigation tends to be large thereby increasing the burden on the user. Our system addresses this problem by selectively showing a subset of the available facet conditions. The suggested conditions are selected based on an intuitive cost model of user's navigation that attempts to minimize the expected navigation actions by hiding uninteresting or ineffective conditions. We provide feasible solutions for this problem and demonstrate their effectiveness by a thorough experimental evaluation and a user study.

11. REFERENCES

[1] S. Abiteboul, R. Hull, V. Vianu: Foundations of Databases. Addison-Wesley 1995.

[2] B. Aditya, G. Bhalotia, S. Chakrabarti, A. Hulgeri, C. Nakhe, Parag, S. Sudarshan: BANKS: Browsing and Keyword Searching in Relational Databases. VLDB 2002: 1083-1086.

[3] Amazon Mechanical Turk. Online: https://www.mturk.com

[4] O. Ben-Yitzhak, N. Golbandi, N. Har'El, R. Lempel, A. Neumann, S. Ofek-Koifman, D. Sheinwald, E. J. Shekita, B. Sznajder, S. Yogev: Beyond Basic Faceted Search. WSDM 2008: 33-44.

[5] M. Bergman. (2000) The Deep Web: Surfacing Hidden Value. [Online] Available: http://brightplanet.com/index.php/white-papers/119.html

[6] K. Chakrabarti, S. Chaudhuri, S. Hwang: Automatic Categorization of Query Results. SIGMOD Conference 2004: 755-766.

[7] S. Chaudhuri, G. Das, V. Hristidis, G. Weikum: Probabilistic Information Retrieval Approach for Ranking of Database Query Results. ACM Trans. Database Syst. 31(3): 1134-1168 (2006).

[8] V. Chvatal: A Greedy Heuristic for the Set Cover Problem. Mathematics of Operations Research 4(3): 233-235 (1979).

[9] W. W. Cohen: Learning Trees and Rules with Set-Valued Features. AAAI/IAAI, Vol. 1 1996: 709-716.

[10] W. Dakka, P. G. Ipeirotis, K. R. Wood: Automatic Construction of Multifaceted Browsing Interfaces. CIKM 2005: 768-775.

[11] W. Dakka, P. G. Ipeirotis: Automatic Extraction of Useful Facet Hierarchies from Text Databases. ICDE 2008: 466-475.

[12] J. Diederich, W. Balke: The Semantic GrowBag Algorithm: Automatically Deriving Categorization Systems. ECDL 2007: 1-13.

[13] J. English, M. A. Hearst, R. R. Sinha, K. Swearingen, K. Yee: Hierarchical Faceted Metadata in Site Search Interfaces. CHI Extended Abstracts 2002: 628-639.

[14] M. A. Hearst: Clustering versus Faceted Categories for Information Exploration. Commun. ACM 49(4): 59-61 (2006).

[15] M. A. Hearst: User Interfaces and Visualization. Modern Information Retrieval. Ricardo Baeza-Yates and Berthier Ribeiro-Neto, Eds. ACM Press, New York, 1999, 257–323.

[16] V. Hristidis, L. Gravano, Y. Papakonstantinou: Efficient IR-Style Keyword Search over Relational Databases. VLDB 2003: 850-861.

[17] V. Hristidis, H. Hwang, Y. Papakonstantinou: Authority-Based Keyword Search in Databases. ACM Trans. Database Syst. 33(1): (2008).

[18] A. Kashyap, V. Hristidis, M. Petropoulos, S. Tavoulari: BioNav: Effective Navigation on Query Results of Biomedical Databases. ICDE 2009: 1287-1290.

[19] Medical Subject Headings http://www.nlm.nih.gov/mesh/

[20] M. Ortega-Binderberger, K. Chakrabarti, S. Mehrotra: An Approach to Integrating Query Refinement in SQL. EDBT 2002: 15-33.

[21] PubMed. http://www.ncbi.nlm.nih.gov/pubmed/

[22] S. B. Roy, H. Wang, G. Das, U. Nambiar, M. K. Mohania: Minimum-Effort Driven Dynamic Faceted Search in Structured Databases. CIKM 2008: 13-22.

[23] N. Sarkas, N. Bansal, G. Das, N. Koudas: Measure-driven Keyword-Query Expansion. PVLDB 2(1): 121-132 (2009).

[24] A. G. Taylor: Wynar's Introduction to Cataloging and Classification, 10th ed. Libraries Unlimited, Inc. 2006.

[25] B. Vélez, R. Weiss, M. A. Sheldon, D. K. Gifford: Fast and Effective Query Refinement. SIGIR 1997: 6-15.

[26] P. Wu, Y. Sismanis, B. Reinwald: Towards Keyword-Driven Analytical Processing. SIGMOD Conference 2007: 617-628.

[27] K. Yee, K. Swearingen, K. Li, M. A. Hearst: Faceted Metadata for Image Search and Browsing. CHI 2003: 401-408.