Top-k Implementation Techniques of Minimum Effort Driven Faceted Search For Databases

Senjuti Basu Roy, Gautam Das

University of Texas at Arlington
Box 19015, 416 Yates St.,Room 300, Nedderman Hall,
Arlington, TX 76019-0015, USA.
senjuti.basuroy@mavs.uta.edu,gdas@uta.edu

Abstract

In this paper, we investigate opportunities to improve the performance of minimum effort driven faceted search techniques. The main idea is motivated by the early stopping techniques used in the TA-family of algorithms for top-k computations. Our initial set of experimental results demonstrate that the proposed techniques expedite performance significantly.

1 Introduction

In recent years, the paradigm of *faceted search* has received significant research attention. In particular, it has been argued that faceted search interfaces [4] can be extremely useful in user navigation and search for data records within data warehouses. Consider an example of a customer care representative in a bank, searching for a particular customer record (e.g., searching for a silver card customer in Texas), but who is only partially familiar with the details of the record. The popular *IR style* techniques of ranked retrieval may not be useful in such cases since the user query is too "generic" and can result in a flood of results. Instead, a faceted search interface may provide an alternative way of searching and navigation in this scenario: first by Branch Location (Texas → Dallas → South Dallas), then drilling down via Age Group (30-50 → 30-35), and then via No of Transactions (Less than 10 transactions in last month → less than 5 debit) and so on. This alternative interface is of great assistance to the customer care representative in successfully locating the record of interest and ending her search.

In recent research works [1, 2] we have proposed techniques to enable effective faceted search for tuples in structured databases. There, the challenge is to determine, from the abundance of available metadata, which attributes of the tuples are best suited for enabling a faceted search interface. For the previous bank database example, a very simple faceted search interface is as follows: the user is prompted an attribute [1] (e.g., Location), to which she responds with a desired value (e.g., Dallas), after which the next appropriate attribute (e.g., Age Group) is suggested to which she responds with a desired value (e.g., 30-35), and so on. Our overall goal is to judiciously select the next facet at every step, so that the user reaches the desired tuple with minimum effort. While the effort expended by a user during a search/navigation session may be fairly complex to measure, we focus on a rather simple but intuitive metric: the expected number of queries that the user has to answer in order to reach the tuple of interest.

Given a database D with m attributes and n tuples, the faceted search algorithms primarily build a decision tree which distinguishes each tuple by selecting attributes (asking questions) that are most likely to lead to the tuple of interest. It is important to note that the attribute selection procedure is not based on traditional decision tree splitting measures such as information gain; rather it is based on the $Indg()$ measure (to be explained later) that attempts to minimize the average height of the tree (the average height is equivalent to the expected number of queries the user has to answer before reaching the tuple of interest). Algorithms are implemented with the help of a scalable decision tree construction technique, leveraging a modification of the Rainforest [5] framework. Using this framework, at every node of a partially built decision tree, a single scan of the database partition associated with that node is sufficient to determine the next best facet. Such an implementation (referred to as the *Full Scan Algorithm* in this paper) achieves a significant speed up over the naive implementation (that requires m complete database scans of the database partition at each node to determine the next best facet).

The main contribution of this paper is to explore

15th International Conference on Management of Data COMAD 2009, Mysore, India, December 9–12, 2009
©Computer Society of India, 2009

[1] Henceforth in this paper facets and attributes will be used interchangeably.

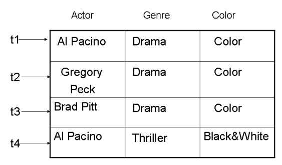
interesting and novel techniques by which the performance of the facet selection algorithms in [1, 2] can be improved even further. To be truly effective, faceted search algorithms have to respond rapidly and without delay during an interactive session with an end user. The Full Scan algorithm presented in [1, 2], while better than any naive strategy, still suffered from high CPU cost and slow response time, as selecting the best attribute at each node required extensive calculations involving the database partition.[2] In this paper, we propose techniques to improve the performance of the facet selection algorithms by reducing CPU intensive computations. The main idea is a novel adaptation of the early stopping techniques used in the TA-family of algorithms for top-k computations [6, 7, 8]. Such techniques can attain early termination that avoid scanning and scoring the complete database in determining the next most promising facet. In addition, as an even faster alternative, we propose an approximate facet selection technique that is guaranteed to stop after reading a fixed number of tuples and return the most promising facet discovered thus far.

2 Preliminaries

In this section, we briefly revisit related technical details of [1, 2]. Essentially, our proposed facet selection algorithms [1, 2] rely on building a minimum cost decision tree [3][3]. Let D be a database with m attributes $\{A_1, \ldots, A_m\}$ and n tuples. Let A_l ($l <= m$) be an attribute of D with $|Dom_l|$ domain values. Picking the attribute A_l as the root node partitions D into disjoint tuple sets $D_{x_1}, D_{x_2}, \ldots, D_{x_{|Dom_l|}}$, where each D_{x_q} is the set of tuples that share the same attribute value x_q of A_l.

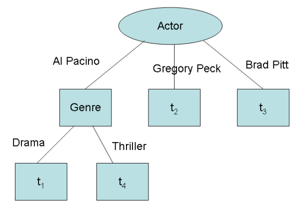
As an example of such decision trees, consider Figure 1(a) which refers to a toy *movie* database with three attributes and four tuples. A decision tree for identifying each of the tuples in the tuple set $D = \{t_1, t_2, t_3, t_4\}$ is shown in Figure 1(b). The leaves of such a decision tree is the tuple set D and each tuple appears exactly once in the leaf nodes. A user reaches her preferred tuple by answering attribute values.

Given such a tree T, $cost(T)$ is defined as $\sum_i ht(t_i)/n$ where $ht(t_i)$ is the height of leaf t_i. Clearly, assuming that each tuple is equally likely to be preferred by the user, this cost represents the average number of queries that needs to be answered before the user arrives at a desired tuple. It is easy to verify



	Actor	Genre	Color
t1	Al Pacino	Drama	Color
t2	Gregory Peck	Drama	Color
t3	Brad Pitt	Drama	Color
t4	Al Pacino	Thriller	Black&White

(a) A small movies database



(b) An optimal decision tree

Figure 1: A Small Movie Database and an Optimal Decision Tree

that the tree in Figure 1(b) is optimal (with minimum cost $= (2 + 2 + 1 + 1)/4 = 1.5$).

Two types of faceted search problems on databases have been considered in [1, 2]: (i) Faceted Search as an Alternative to Ranked-Retrieval and (ii) Faceted Search that Leverages Ranking Functions (henceforth referred to as FSNoRank and FSRank respectively in this paper). Essentially the second problem assumes that a ranking/scoring function is available that describes the preferences of the user for each tuple in a partition (see [1, 2] for more details).

For FSNoRank, attribute A_l is selected as the root of the decision tree if it minimizes the $Indg(A_l, D)$ (over all attributes of D) value. Formally, $Indg(A_l, D)$ for FSNoRank can be expressed as

$$Indg(A_l, D) = \sum_{1 \le q \le |Dom_l|} |D_{x_q}|(|D_{x_q}| - 1)/2 \quad (1)$$

The intuition is that any decision tree should distinguish every pair of distinct tuples. The approach is to make the attribute that distinguishes the maximum number of pairs of tuples as the root of the tree, where an attribute A_l is said to distinguish a pair of tuples t_i, t_j if $t_i[l] \neq t_j[l]$. We note that these definitions of $Indg()$ are different from the standard measures used for splitting attributes in decision trees, such as information gain.

Variants of FSNoRank and FSRank are also considered in the paper [1, 2], i.e., we have presented algorithms for computing single facet at every step, extending that to k-facets selection and a fixed k-facets interface. We refer to the papers [1, 2] for more details.

We also assume that attributes are associated with *uncertainties*, where the uncertainty of an attribute refers to the probability of the user being able to provide a desired value for that attribute. Intuitively this means, a customer service representative of a bank searching for a specific customer may not know exactly the street address of the customer's residence; likewise a user searching for a movie may not be sure of the director of the desired movie, and so on. In designing our decision trees to cope with uncertainty, we assume that users can respond to a question by either (a) providing the correct value of the queried attribute A_i, or

[2]The I/O cost is typically not an issue, as with the latest advent of semiconductor technology, even an inexpensive personal computer can often store an entire database partition associated with a decision tree node in main memory. It is the computational costs that are more critical to attain real-time response during interactions with an end user.

[3]Briefly, each node of the tree represents an attribute, and each edge leading out of the node is labeled with a value from the attributes domain.

(b) responding with a "don't know". In either case, the faceted search system has to respond by questioning the user with a fresh attribute. The inability of the user to always respond with the correct attribute value raises interesting complications in the design of appropriate decision trees.

Therefore, in the constructed decision tree, each node A_l now has $|Dom_l| + 1$ links, with one of the links labeled as "don't know". This link is taken with probability $1 - p_l$ when the user does not know the value of this attribute, whereas the rest of the links are taken with probability p_l when the user knows the value of the attribute. Thus, in the former case, the attribute A_l cannot distinguish any further pairs of tuples (the query was essentially wasted), whereas in the latter case, the attribute manages to distinguish several pairs of tuples and only $Indg(A_l, D)$ pairs were left indistinguished. Thus, we can see that if we select A_l as the root node, then the *expected* number of tuple pairs that cannot be disambiguated is $(1 - p_s) \times |D|(|D| - 1)/2 + p_s \times Indg(A_l, D)$. It is not hard to see that an obscure attribute that has little chance of being answered correctly by most users, but is other very effective in distinguishing attributes, will be overlooked in favor of other attributes in the decision tree construction.

For FSRank, we additionally have a scoring (or ranking) function. Let the scoring function assign a score of $S(Q, t)$ to a tuple t in D_{x_q}, where Q is the current query that selects the partition. As discussed in [1, 2], this may be interpreted as the probability with which the tuple is the target tuple in the partition. $Indg(A_l, D)$ for FSRank can be expressed as

$$Indg(A_l, D) = \sum_{x_q \in Dom_l} \left(\sum_{t_i, t_j \in D_{x_q}, i<j} S(Q, t_i) \times S(Q, t_j) \right) \quad (2)$$

Given the above discussion, the cost of a specific decision tree T becomes more complicated than the corresponding definition when no ranking function was assumed. Consider a database D selected by an initial query Q, and consider a decision tree T with each tuple of D at its leaves. We will thus derive a formula for $cost(T, Q)$. Note that Q needs to be a parameter in the cost, as the ranking function uses Q to derive preference probabilities for each tuple. Note that in this cost definition we are not considering attribute uncertainties.

Let the root of the tree select the facet A_l. The root partitions D into the sets $D_{x_1}, \ldots, D_{x_{|Dom_l|}}$ where D_{x_q} is the set that satisfies the query $Q \wedge (A_l = x_q)$ for each $x_q \in Dom_l$. Let the corresponding subtrees for each of these partitions be $T_{x_1}, \ldots, T_{x_{|Dom_l|}}$.

Clearly $cost(T_{x_q}, Q \wedge (A_l = x_q))$ is the (recursive) cost of each subtree. The quantity $\sum_{t \in D_{x_q}} S(Q, t)$ is the cumulative probabilities of all tuples in D_{x_q} and represents the probability that when the user is at the root, she will prefer any of the tuples in D_{x_q}. Thus we have

$$cost(T, Q) = \sum_{x_q \in Dom_l} \left(\sum_{t \in D_{x_q}} S(Q, t) \right) \times \big(cost(T_{x_q}, Q \wedge (A_l = x_q)) + 1\big) \quad (3)$$

It is easy to see that if no ranking functions are assumed, i.e., each tuple is uniformly preferred by the user, the cost of a tree reduces to the definition in Section 2, i.e., $\sum_{t \in D} ht(t)/n$.

Our task is then the following: Given an initial query Q that selects a set of tuples D, to determine a tree T such that $cost(T, Q)$ is minimized.

3 Efficient Facet Selection Techniques

In this section, we focus on improving response time of the facet selection algorithms, by leveraging early stopping techniques from top-k algorithms.

3.1 Exact Indg() Calculation Based On Top-k Computation

In general, top-k algorithms operate on index lists corresponding to a query's elementary conditions and aggregate scores monotonically for result candidates. The objective is to terminate the index scans as early as possible based on lower and upper bounds for the scores of result candidates. Motivated by such early stopping techniques employed in top-k algorithms, we wish to determine the best facet (or the best set of k facets) at every step of faceted navigation without performing a complete database partition scan.

In this paper, we assume that the ranking function in the FSRank problem is accessible via a *pipelining interface*, which is natural and supported by previous works on top-k computation such as [6, 7, 8]. The *pipelining interface* $S(Q, D)$ takes as input a query Q and a database D and outputs a stream of tuples ranked descending according to $S(Q, t)$ along with their scores. The cost incurred in using this interface is the number of tuples retrieved (we can stop retrieving tuples at any time).

The high-level idea of early stopping is as follows: while scanning the database partition D_1, we consume tuples in some sequence and maintain a lower and upper bound for the value of $Indg(A_s, D_1)$ for each attribute A_s, and stop as soon as we discover an attribute A_l whose upper bound is no larger than the lower bound of all other attributes.

Lemma 3.1. *Given a database D with n tuples and m attributes, a CPU speedup of $n/(r+(n-r)/m)$ over the*

Full Scan algorithm can be obtained, if only r ($r \leq n$) tuples are consumed before the next best facet can be determined.

Proof Sketch: The Rainforest implementation of Full Scan requires $n \times m$ update operations to compute $Indg()$ and return the best facet to the user (along with its domain information). Using the pipelining interface, if the best facet is determined after reading r tuples, then the total number of update operations required to suggest the best facet to the user (along with its domain information) is $(r \times m) + (n - r)$, where the first term refers to the update cost of processing the first r tuples, and the second term refers to the update cost of processing the remaining tuples, where only the counts for the selected attribute are updated. Therefore the speedup is $n/(r + (n - r)/m)$. □

As an illustrative example, for a database partition containing $200k$ tuples and 50 attributes, if only 20% of the tuples are consumed before termination, then the CPU speedup over Full Scan is $200k/(40k + 160k/50) = 4.6$.

We next discuss the FSRank case in detail. Assume that the pipelined interface has already scored r tuples, and let D_r be the set of tuples with the highest scores. Let the score of the rth tuple be S_r. For each attribute A_s, we maintain the following two quantities:

$$LowerIndg(A_s, D) = Indg(A_s, D_r) \quad (4)$$

$$\begin{aligned} UpperIndg(A_s, D) = {} & \\ & LowerIndg(A_s, D) + \\ & (n-r)S_r \times \max_{x_q \in Dom_s} \left\{ \sum_{t \in D_r, t[s] = x_q} S(Q,t) \right\} + \\ & S_r \times S_r \times (n-r)(n-r-1)/2 \end{aligned} \quad (5)$$

The lower bound of $Indg()$ reflects the minimum value that attribute A_s can get, i.e., it assumes that the rest $(n - r)$ tuples will not contribute anything to the $Indg()$ value. This implies that each tuple that is not read yet has a unique domain value under attribute A_s. Therefore, $LowerIndg(A_s, D) = Indg(A_s, D_r)$.

On the other hand, the upper bound of $Indg()$ captures the maximum cumulative value attribute $Indg(A_s, D)$ can attain from the rest $(n - r)$ tuples by considering that the rest $(n - r)$ tuples have the same score S_r and can be paired with the largest subset of already read r tuples with the same domain value. This implies, if x_q is the largest domain value of attribute A_s so far, then the domain value of attribute A_s for the rest $(n - r)$ tuples is also x_q. The $UpperIndg(A_s, D)$ formula contains the sum of two terms. The first term captures the value obtained by pairing each $(n - r)$ tuples with each tuple in the largest subset of r tuples with domain value x_q, whereas the second term captures the value accumulated by pairing each $(n - r)$ tuples with each other.

The pipelining interface consumes tuples and maintains these bounds, and stops when it discovers an attribute A_l whose upper bound is no larger than the lower bound of all other attributes. We refer to this as Exact FSRank Algorithm.

Algorithm 1 Eaxct FSRank (D, A')

```
1: Input: a database $D$ with $n$ tuples, a subset $A' \subset A$ attributes not yet used
2: Output: Attribute $A_1 \in A'$ that minimizes the $Indg()$ value.
3: begin
4: **if** $|D| = 1$ **then**
5:     Return a tree with any attribute $A_l \in A'$ as a singleton node
6: **end if**
7: **if** $|A'| = 1$ **then**
8:     Return a tree with the attribute in $A'$ as a singleton node
9: **end if**
10: Read the first tuple
11: Set $r = 2$
12: **while** $r <= n$ **do**
13:     Read the $r$-th tuple
14:     **for** each $A_1 \in A'$ **do**
15:         $LowerIndg(A_1, D) = Indg(A_1, D_r)$
16:         $UpperIndg(A_1, D) = LowerIndg(A_1, D) + (n-r)S_r \times \max_{x_q \in Dom_s} \left\{ \sum_{t \in D_r, t[s]=x_q} S(Q,t) \right\} + S_r \times S_r \times (n-r)(n-r-1)/2$
17:     **end for**
18:     $ChosenAttribute = \operatorname{argmin}_{A_1 \in A'} UpperIndg(A_1, D)$
19:     **for** each $A_i \in A' - \{ChosenAttribute\}$ **do**
20:         **if** $UpperIndg(ChosenAttribute, D) \leq LowerIndg(A_i, D)$ **then**
21:             Continue
22:         **else**
23:             $r = r + 1$
24:             Return to the while loop
25:         **end if**
26:     **end for**
27:     return $ChosenAttribute$
28: **end while**
29: end
```

The Exact FSNoRank Algorithm is very similar, except that the upper and lower $Indg()$ for each attribute A_s is computed as follows:

$$LowerIndg(A_s, D) = Indg(A_s, D_r) \quad (6)$$

$$UpperIndg(A_s, D) = LowerIndg(A_s, D) + [(n-r) + \max_{x_q \in Dom_s} \{D_{x_q}\}]/2 \left\{(n-r) + \max_{x_q \in Dom_s} \{D_{x_q}\} - 1\right\} \quad (7)$$

Here the pipelining interface outputs tuples in any arbitrary order (since there is no ranking function), and stops when it discovers an attribute A_l whose upper bound is no larger than the lower bound of all other attributes. We refer to this as the Exact FSNoRank Algorithm.

The $LowerIndg(A_s, D)$ calculation in the FSNoRank case has a similar explanation as for the FSRank case, except for the score of each tuple is assumed to be 1 here. Also, the score of each tuple is assumed to be 1 in the $UpperIndg(A_s, D)$ calculation, and the upper bound simply captures the $Indg()$ value that can be attained by pairing each of the rest $(n-r)$ tuples with the largest subset of r tuples with domain value x_q.

Although our attribute selection algorithms can work for any black box scoring function $S(Q, t)$, the score distribution across the tuples greatly affects the performance of our algorithms, since it determines which algorithms are feasible and efficient. A highly skewed scoring function - where the top few tuples have large scores, followed by a rapid degradation in score values for the remaining tuples - is most effective in making the Exact FSRank algorithm very efficient. This unfortunately does not apply in the case of Exact FSNoRank, because there is no ranking function to be leveraged. In fact as the lemma below shows, no matter what is the database, more than half of the database has to be *always* scanned by the FSNoRank algorithm before the best attribute can be determined.

Lemma 3.2. *For FSNoRank, even in the best case more than half of the database partition has to be scanned using the pipelining interface before the best facet can be determined.*

Proof Sketch: Consider a simple case, where a database D with n (n is even) tuples has only two attributes A and B. Let us assume that already $n/2$ tuples have been read, and the best scenario of early stopping has occurred so far in D, i.e., attribute A has returned $n/2$ different domain values $a_1, a_2, \ldots, a_{n/2}$, while attribute B repeats the same domain value b_1 in all $n/2$ tuples. Then we have, $LowerIndg(A, D) = 0$ and $UpperIndg(A, D) = n(n+2)/8$. Similarly, $LowerIndg(B, D) = n(n-2)/8$ and $UpperIndg(B, D) = n(n-1)/2$.

At this stage, no stopping decision can yet be made considering the upper and lower bounds of the $Indg()$ values of the attributes, and we must continue the scan of D. □

3.2 Approximate Indg() calculation

As an even faster alternative to the above algorithms, we can simply stop reading further tuples after a small fixed number of iterations (i.e., bounded r), and use the most promising facet discovered thus far. Such an algorithm is of course guaranteed not to exhaust all tuples in the database partition, but may not necessarily produce the facet with the minimum $Indg()$ value. However, this is a good approximation if r is reasonably chose. It is easy to observe that such an approximate $Indg()$ calculation can be applied to both FSNoRank and FSRank.

4 Experimentation and Results

In this section we describe experimental evaluations and draw conclusions on the effectiveness of our proposed techniques.

Hardware: All experiments are run on a machine having Intel(R)Xeon(TM) CPU with 3.0 Ghz processor and 2.0 GB RAM running Windows XP. All algorithms are implemented using Java and $C\#$. T-SQL is used to query the back end SQL Server 2005 database management system.

Database Used: We run all our experiments on a subset of the Yahoo Autos dataset, a nationwide online used-car automotive dealer's database. Our experimental dataset contains 1 million tuples and 43 derived attributes.

Ranking Function: Design of an efficient and effective ranking function is an orthogonal research problem and is not our focus here. For practical purposes however, we implement a simple *Squared Distance* based ranking function. In this Squared Distance function, a tuple t gets a score equal to the square of its Euclidian Distance from the centroid of the residual database partition. We further normalize this squared distance to a non-uniform probability distribution over the selected tuples, such that $S(Q, t)$ represents the probability that tuple t is preferred by the user, and that $\sum_{t \text{ selected by } Q} S(Q, t) = 1$.

Performance Evaluation Performance is measured in terms of the average node creation time. In this case, we vary database size and observe the performance of three different algorithms. Performance is evaluated among the FSRank Full Scan Algorithm, the Exact FSRank Algorithm and the Approximate FSRank ($r = 100$) Algorithm.

Figure 2 corroborates our claim - the average node creation time can be significantly improved in Exact FSRank Algorithm compared to the FSRank Full Scan Algorithm. The Approximate FSRank is the fastest, but it comes with a loss in quality - the navigation cost is sometimes more.

Change of cost by varying r: We vary the parameter r here - which determines how many tuples are to

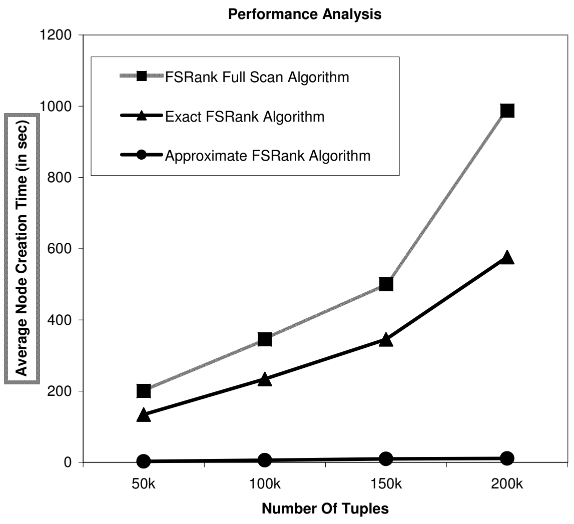


Figure 2: Average Node Creation Time Varying Dataset Size

be read before the pipelining interface is terminated to make the selection of the attribute. As expected, by deciding r in advance, we lose quality (i.e., increase the average navigation cost) as a trade off to the performance. However, the navigation cost decreases as r increases. An interesting problem here can be to find an optimal r value for a given dataset. This concludes

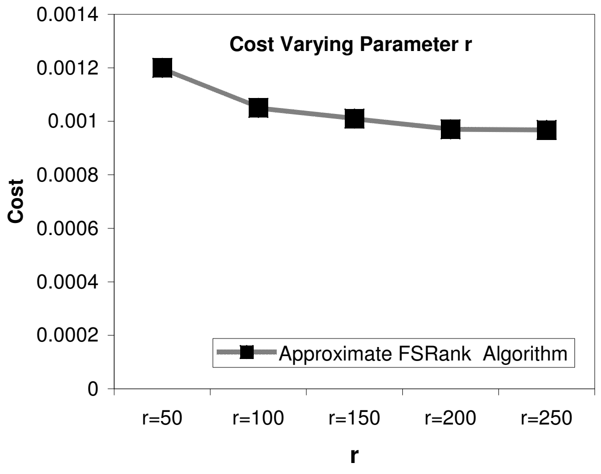


Figure 3: Change of Cost Varying r in Approximate FSRank Algorithm

our discussion on experiments.

5 Ongoing and Future Work

In this paper, we propose techniques to reduce CPU intensive costs during minimum effort driven faceted navigation. As ongoing work, we aim to extend these techniques to design efficient algorithms for other types of decision trees. For example, it would be interesting to see if traditional decision tree algorithms, designed for classification purposes, can benefit from such early stopping techniques.

We also aim to explore other techniques, such as sampling, that can assist in expediting the response time of facet selection algorithms. Such techniques may be useful in approximating domain information of the attributes in a principled way, thus guaranteeing a reduced CPU cost while suggesting facets to the user. We would also like to perform a comparative quality evaluation of these various proposed techniques on a variety of real world datasets. In the future, we would like to conduct user studies to obtain user evaluations on our proposed speedup techniques.

References

[1] S. Roy, H. Wang, G. Das, U. Nambiar and M. K. Mohania. Minimumeffort driven dynamic faceted search in structured databases. CIKM Conference, 13–22, 2008.

[2] S. Roy, H. Wang, G. Das, U. Nambiar and M. K. Mohania. DynaCet: Building Dynamic Faceted Search Systems over Databases. ICDE Conference, 1463-1466, 2009.

[3] Venkatesan. T.Chakravarthy, Vinayaka. Pandit, Sambudha. Roy, Pranjal. Awasthi and Mukesh. Mohania. Decision Trees for Entity Identification: Approximation Algorithms and Hardness Results *ACM PODS*, 53–62, 2007.

[4] Jennifer. English, Marti. Hearst, Rashmi. Sinha, Kirsten. Swearingen and Ping. Yee. Hierarchical Faceted Metadata in Site Search Interfaces. *CHI Conference Companion*, 628–639, 2002.

[5] Johannes. Gehrke, Raghu. Ramakrishnan and Venkatesh. Ganti. *RainForest - A Framework for Fast Decision Tree Construction of Large Datasets. s. Data Min. Knowl. Discov.*, 4(2/3): 127–162, 2000.

[6] R. Fagin. *Combining Fuzzy Information from Multiple Systems.* PODS, pages 216226, June 1996.

[7] U. Guntzer and W.-T. Balke and W. Kiesling. *Optimizing multi-feature queries for image databases.* The VLDB Journal, pages 419428, 2000

[8] R. Fagin and A. Lotem and M. Naor. *Optimal Aggregation Algorithms For Middleware.* PODS, June 2001.