Context-sensitive ranking

Rakesh Agrawal[*]
Microsoft Search Labs
Mountain View, CA, USA
rakesh.agrawal@microsoft.com

Ralf Rantzau
IBM Almaden Research Center, San Jose, CA, USA
rrantza@us.ibm.com

Evimaria Terzi [*]
HIIT Basic Research Unit
University of Helsinki, Finland
terzi@cs.helsinki.fi

ABSTRACT

Contextual preferences take the form that item i_1 is preferred to item i_2 in the context of X. For example, a preference might state the choice for Nicole Kidman over Penelope Cruz in drama movies, whereas another preference might choose Penelope Cruz over Nicole Kidman in the context of Spanish dramas. Various sources provide preferences independently and thus preferences may contain cycles and contradictions. We reconcile democratically the preferences accumulated from various sources and use them to create a priori orderings of tuples in an off-line preprocessing step. Only a few representative orders are saved, each corresponding to a set of contexts. These orders and associated contexts are used at query time to expeditiously provide ranked answers. We formally define contextual preferences, provide algorithms for creating orders and processing queries, and present experimental results that show their efficacy and practical utility.

1. INTRODUCTION

The curse of abundance has arrived with vengeance for database-centric web applications. The online comparison site, Bizrate.com, now carries more than 30 million product offers from more than 40,000 stores. Another site, shopping.com, carries more than 1.5 million product offers in the category of home furnishing alone. The conventional query-processing technology that simply returns the set of tuples satisfying the query predicate is woefully inadequate for such applications. These applications require the result tuples to be ranked, factoring in the context of the query, without sacrificing the query performance.

We present a solution framework for addressing the above problem, comprising of the following components:

1. Use of mass collaboration to accumulate contextual preferences of the form $i_1 \succ i_2 \mid X$, meaning that item i_1 is preferred to item i_2 in the context of X.
2. Techniques for using these preferences to generate a few representative orderings of the tuples and their associated contexts.
3. Techniques for using these pre-orderings to quickly provide ranked answers to the queries taking into consideration the condition part of the query.

EXAMPLE 1: Consider the *movie* relation with schema (*title*, *actor*, *genre*, *language*). Assume that we have the following contextual preferences (written omitting attribute names):

1. Nicole Kidman $\succ$ Penelope Cruz $\mid$ Drama
2. Penelope Cruz $\succ$ Nicole Kidman $\mid$ Drama and Spanish
3. Daniel Brühl $\succ$ Nicolas Cage $\mid$ German

These preferences illustrate that the ranking of the tuples of a relation is subjective. Nicole Kidman has played in many more movies than Penelope Cruz and has been nominated for many more academy awards. However, in the context of Spanish dramas the latter actress is considered more important. Similarly, for Nicolas Cage and Daniel Brühl. Cage has much richer movie repertoire. However, in the context of German movies, Brühl can be considered more important. This example shows that one cannot rank objects independently of the context in which they appear. Now, consider the following query:

```
SELECT actor
FROM   movie
WHERE  genre = 'Drama' and
       language = 'Spanish'
```

Since the WHERE clause of the query contains the set {drama, Spanish}, the preferences that refer to either or both drama and Spanish, i.e. preferences 1 and 2, should be taken into consideration in the ranking of the query results. □

1.1 Problem

We address the following problem: Given a database table r and a set of contextual preferences $\mathcal{P}$, incorporate the knowledge of $\mathcal{P}$ into the query-answering mechanism. More specifically, for a given query, the contextual preferences that are related to it are taken into consideration in order to provide ranked top-k results.

For the purposes of this paper, we will assume that the set of contextual preferences $\mathcal{P}$ is available to us. In practice, this set will

[*]Work done while the author was at the IBM Almaden Research Center.

be collected using various forms of mass collaboration. There may be volunteer users willing to provide preferences much in the same way users voluntarily rate products and services on the web. They may be automatically collected by observing the selections of willing users with the help of browser plug-ins or taskbars. In some domains, these preferences can be learned from past data. In fact, we employed association-rule mining to generate preferences from the internet movie database for use in our experiments. We admit conflicts and cycles in the user-provided preferences and resolve them democratically.

1.2 Related Work

There is marked similarity between our framework and the approach taken by web-search engines to handle the abundance of web pages. Much like Google's page rank [8], we "rank" tuples a priori, independent of a specific query, and then use the information in the query and the saved "rankings" to quickly provide ranked answers. The use of contexts also bears resemblance to personalized page rank [21, 32]. However, while in the case of the web, the hyperlinks between the pages define a natural graph structure that yields a good ranking of the web pages, there is generally no corresponding meaningful structure in the relational data. We, therefore, use a graph induced by user-provided preferences between tuples for this purpose. Additionally, unlike web searches where all queries are essentially of the same form (Boolean expressions over terms), the database queries are much more diverse, which has a bearing on the specific techniques employed.

There has been fruitful research on inducing a graph structure based on the contents of tuples of a database. For instance, in [6], tuples constitute the nodes, connected by edges induced by foreign key and other relationships. Answers to keyword queries are ranked using a notion of proximity coupled with the prestige of nodes based on incoming edges. Other examples of efforts in this direction include [4, 20, 23]. Although very interesting, this line of work further reinforces that while in the case of the web the structure of the web graph is apparent, there is no globally accepted graph structure that can represent relational data.

There is also work on defining the importance of a tuple in a relation via user-expressed numeric preferences and using them to rank query results [2]. The skyline operator [7, 31] also considers numeric preferences to output only the interesting query results. A result is considered interesting if it dominates the constants appearing in the query predicates. In [3, 9, 19], the importance scores are extracted automatically by mining past workloads, that can reveal what users are looking for and what they consider as important. We do not have a workload mining step, but we do admit the possibility of mining contextual preferences from the data. We then incorporate pairwise preferences into the formulation of final rankings in a combinatorial manner.

The problem of incorporating personal preferences to create personalized rankings of database query results has been studied in [28]. Atomic user preferences are expressed via simple predicates. Each such predicate is associated with a numeric value that corresponds to the degree of interest of the user to this specific preference. Complex preferences are expressed via composition of atomic ones. The result is a user preference profile. The preferences expressed in the profile are taken into account when answering a query. More specifically, for query q, the subset of user preferences related to q is identified, and is used to alter the query via a rewriting mechanism. This basic model has been enhanced in [27, 29]. In [27] the ranking of query results not only depends on the user's preferences but also on other constraints like the execution cost or the size of its output. Our approach and preference language are very different. Our focus is not on personalizing answers for individual users, but on using a common set of preferences for efficiently answering and ranking query results of a large number of users.

There is a rich body of research on the properties and the expressive power of preference languages. For instance, the work in [25] proposes a generic model for expressing preferences. The model defines some basic preference constructors and more complex preferences are built by their composition. The framework admits preferences that are partial orders. In [10], preferences are specified as first-order formulas. Arbitrary operations on preference formulas are allowed. An operator (the *winnow* operator) is defined so that complex preference formulas are embedded into relational algebra. Query-rewriting mechanisms for doing this are given. In this case, the tuples are output in such a way that no preference constraint is violated. The focus of our work is different, since we formulate the contextual preference satisfaction as an optimization problem. We do not restrict the nature of input preferences, and we allow outputs that do not satisfy all the input preferences but agree with them as much as possible.

Closely related is the work reported in [12] wherein simple pairwise preferences over a set of objects are considered and a total ordering of the input objects that agrees as much as possible with the given preferences input is extracted. We make use of some of these ideas, but enhance the preferences with contexts and focus on how the preferences associated with different contexts have impact on the query results. Finally, we note that our preference language bears resemblance to the triples of training data of the form "with respect to object c, object a is closer than b" used for learning distances in the clustering applications discussed in [30, 33].

1.3 Roadmap

The rest of the paper is structured as follows. In Section 2 we formally define contextual preferences. In Section 3 we define the problem addressed and its decomposition into three subproblems. The algorithmic solutions for each one of them are discussed in Sections 4, 5 and 6. The impact of tuples for which preferences have not been explicitly expressed is discussed in Section 7. The experimental evaluation of our proposal is given in Section 8. We conclude with a summary and directions for future work in Section 9.

2. CONTEXTUAL PREFERENCES

Consider a database relation r with n tuples $r = \{t_1, \ldots, t_n\}$ with schema $R(A_1, \ldots, A_d)$. Let $\mathrm{Dom}(A_i)$ represent the *active domain* of attribute A_i.

Contextual preferences, or simply preferences, are of the form $\{A_i = a_{i1} \succ A_i = a_{i2} \mid X\}$, where X is $\bigwedge_{j \in l} (A_j = a_j)$, with $a_{i1}, a_{i2} \in \mathrm{Dom}(A_i)$, $l \subseteq \{1, \ldots, d\}$ and $a_j \in \mathrm{Dom}(A_j)$. The left-hand side of a preference specifies the *choice* while the right-hand side is the *context*. The semantics of such a contextual preference in terms of the database tuples is the following: all tuples $t \in r$ such that $t.A_i = a_{i1}$ and $\forall j \in l$: $t.A_j = a_j$ are *preferred* to tuples $t' \in r$ for which $t'.A_i = a_{i2}$ and $\forall j \in l$: $t'.A_j = a_j$. Note that our preferences are set-oriented; a single preference can specify choices between a large number of tuples.

EXAMPLE 2: Consider the toy relation of Table 1 with schema

	A_1	A_2	A_3	A_4
t_1	a	β	x	u
t_2	a	γ	y	u
t_3	b	δ	w	u
t_4	c	ϵ	y	u
t_5	a	γ	z	v

Table 1: Toy relation

$R(A_1, A_2, A_3, A_4)$ and the following preferences for tuples in this relation:

$$p_1 = \{A_1 = a \succ A_1 = b \mid A_4 = u\}$$

$$p_2 = \{A_2 = \beta \succ A_2 = \delta \mid A_4 = u\}$$

$$p_3 = \{A_3 = w \succ A_3 = x \mid A_4 = u\}$$

$$p_4 = \{A_1 = c \succ A_1 = a \mid A_3 = y \wedge A_4 = u\}$$

Preference p_1 suggests that in the context of "$A_4 = u$", tuples t_1 and t_2 are preferred to tuple t_3. In the same context, from p_2, tuple t_1 is preferred to t_3, and from p_3 tuple t_3 is preferred to t_1. Finally, from p_4, tuple t_4 is preferred over tuple t_2 in the context of "$A_3 = y$" and "$A_4 = u$". □

For any single preference p and any pair of tuples (t_i, t_j), p either prefers t_i to t_j (denoted by $t_i \succ_p t_j$) or t_j to t_i (denoted by $t_i \succ_p t_j$) or it is inapplicable with respect to t_i and t_j (denoted by $t_i \sim_p t_j$). Thus, every preference p defines PREF over any pair of tuples t, t' that evaluates as follows:

$$\text{PREF}(t, t', p) = \begin{cases} 1 & \text{if } t \succ_p t', \\ 0 & \text{if } t' \succ_p t, \\ \perp & \text{if } t \sim_p t'. \end{cases}$$

For any two contexts $X_1 = \bigwedge_{j \in l_1} (A_j = a_j)$ and $X_2 = \bigwedge_{j \in l_2} (A_j = b_j)$ with $l_1, l_2 \subseteq \{1, 2, \ldots, d\}$, we say that they are equal if and only if $l_1 = l_2 = l$ and $a_j = b_j$ for all $j \in l$. We say that two preferences $\{A_i = a_{i1} \succ A_i = a_{i2} \mid X_1\}$ and $\{A_j = a_{j1} \succ A_j = a_{j2} \mid X_2\}$ belong to the same *preference class* if $X_1 = X_2 = X$. We use P_X to denote the set of all preferences in the same class defined by context X. For example, preferences p_1, p_2 and p_3 from Example 2 belong to the same class defined by context "$A_4 = u$", while p_4 belongs to a different class.

The set of preferences P_X in a class, partitions the tuples in the relation in two sets, the set of *indifferent* and the set of *asserted* tuples. A tuple t is called *indifferent* with respect to a context X if $\forall p \in P_X$ it holds that: $\forall t' \neq t$, $\text{PREF}(t, t', p) = \perp \wedge \text{PREF}(t', t, p) = \perp$. In other words, a tuple is indifferent with respect to a context if no explicit preferences that involve it have been expressed within the context. All tuples that are not indifferent are *asserted*. Intuitively, a tuple is asserted if there exists one or more preferences within a context that explicitly compare this tuple against another tuple of the relation.

Lets consider again preferences p_1, p_2 and p_3 from Example 2. These preferences are all in the same class defined by context "$A_4 = u$". With respect to this class, tuples t_1, t_2 and t_3 are asserted since they are explicitly compared to each other via one of the three preferences of the class. Tuples t_4 and t_5 however are indifferent with respect to this class since there does not exist a preference in the class that compares neither of those two tuples against other tuples of the relation.

The set of preferences of a class defines the *effective preference* (EFF-P) for ordered pairs of tuples (t, t'): For any ordered pair of asserted tuples (t, t') such that there exists a $p \in P_X$ for which $\text{PREF}(t, t', p) = 1 \vee \text{PREF}(t', t, p) = 1$ we have that:

$$\text{EFF-P}(t, t', P_X) = \frac{\sum_{p \in P_X} \text{PREF}(t, t', p)}{\sum_{p \in P_X} (\text{PREF}(t, t', p) + \text{PREF}(t', t, p))}.$$

If for a pair of asserted tuples (t, t') there does not exist a $p \in P_X$ for which $\text{PREF}(t, t', p) = 1 \vee \text{PREF}(t', t, p) = 1$, then

$$\text{EFF-P}(t, t', P_X) = \text{EFF-P}(t', t, P_X) = \frac{1}{2}.$$

If for a pair of tuples (t, t'), t or t' (or both) are indifferent with respect to context X, then: $\text{EFF-P}(t, t', P_X) = \perp$.

Let us go back to Example 2 and consider the preference class $P_{A_4=u}$, which we denote by P_u for brevity. For this preference class, the effective preferences for the tuples are as follows:

$\text{EFF-P}(t_1, t_2, P_u) = \text{EFF-P}(t_2, t_1, P_u) = \frac{1}{2}$, since both t_1 and t_2 are asserted tuples, but no preference has been expressed between them.

$\text{EFF-P}(t_1, t_3, P_u) = \frac{1+1}{3} = \frac{2}{3}$, $\text{EFF-P}(t_3, t_1, P_u) = \frac{1}{3}$, since two preferences in the class prefer t_1 to t_3, but only one prefers t_3 over t_1.

$\text{EFF-P}(t_2, t_3, P_u) = 1$ and $\text{EFF-P}(t_3, t_2, P_u) = 0$.

$\text{EFF-P}(\cdot, \cdot, P_u) = \perp$ for all other pairs of tuples.

Notice that for every pair of asserted tuples (t, t') and every context X the value of $\text{EFF-P}(t, t', P_X)$ is a value in $[0, 1]$ and has a probability interpretation. It corresponds to the probability that there exists a preference $p \in P_X$ as part of the input such that $t \succ_p t'$. In other words, it is the probability of a user preferring t to t' among the users that have provided preferences. When such a preference does not exist, the probabilities of the events "t is preferred to t'" and "t' is preferred to t" are equal (and thus equal to $\frac{1}{2}$).

For a given class of preferences P_X and a relation r, we define the X-preference graph $G_X(V_X, E_X)$ as follows: The set of nodes is the set of all asserted tuples in r. For every ordered pair of nodes (t, t') there exists a directed edge $e(t \to t') \in E_X$, with weight:

$$w_X(t \to t') = \text{EFF-P}(t, t', P_X), \qquad (1)$$

and $w_X(t \to t') + w(t' \to t) = 1$.

Figure 1 shows the preference graph for $P_{A_4=u}$ for the toy relation and preferences p_1, p_2 and p_3 given in Example 2. Note that tuples t_4 and t_5 do not appear in the graph as they are indifferent.

In order to simplify exposition, we will assume in the next four sections that there are no indifferent tuples in the relations we are considering. The implications of indifferent tuples to our proposed algorithms are discussed in Section 7.

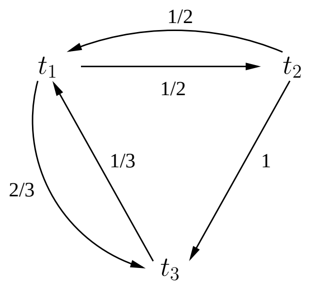

Figure 1: Preference graph for context "$A_4 = u$"

3. THE RANK SELECTION PROBLEM

Consider a conjunctive selection query q over relation $r = \{t_1, \ldots, t_n\}$ with schema $R(A_1, \ldots, A_d)$. Let the active domain of attribute A_i be $\text{Dom}(A_i)$. The query is of the form $q = \sigma_{\wedge_{j\in\{1,\ldots,u\}}}(A_{i_j} = a_j)$, where $a_j \in \text{Dom}(A_{i_j})$. Let $q(r) \subseteq r$ be the subset of tuples in r that are in the answer of q. The goal of the paper is to address the RANK SELECTION PROBLEM defined as follows:

PROBLEM 1 (RANK SELECTION PROBLEM): Assume a set of preferences $\mathcal{P} = \{P_{X_1}, \ldots, P_{X_m}\}$ and a selection query q. The rank-selection problem asks for a permutation of τ of $q(r)$ such that

$$\tau = \arg\max_{\tau'} \sum_{i=1}^{m} \text{sim}(q, X_i)\textsc{Agree}(\tau', P_{X_i}),$$

where

$$\textsc{Agree}(\tau, P_X) = \sum_{(t,t'):\tau(t)<\tau(t')} \textsc{Eff-P}(t, t', P_X).$$

The intuition of the problem formulation is that we want to find the permutation τ over the set of tuples in $q(r)$ that agrees as much as possible with the input preferences. Additionally, the degree of agreement with a class of preferences is weighted by the similarity between the contexts of those preferences and the imposed query q.

For an adequate definition of Problem 1 we need to quantify the similarity between a query and a context. We adopt a typical definition of similarity, the *cosine similarity*. For this to be defined we first need to form the vector representations of a context and a conjunctive selection query.

Consider the set $\mathcal{D}$ of all distinct ⟨attribute, attribute-value⟩ pairs appearing in the r, that is, $\mathcal{D} = \{\langle A_i, a\rangle \mid \forall i \in \{1, \ldots, d\}$ and $\forall a \in \text{Dom}(A_i)\}$. Since $\text{Dom}(A_i)$ is the active domain of attribute A_i the cardinality of this set is finite. Let it be $N = |\mathcal{D}|$ and let O_D be an arbitrary but fixed order on the pairs appearing in $\mathcal{D}$. We refer to the i-th element of $\mathcal{D}$ based on the ordering O_D by $\mathcal{D}[i]$. A vector representation of a context $X = \bigwedge_j (A_j = a_j)$ is a binary vector V_X of size N. The i-th element of the vector corresponds to pair $\mathcal{D}[i]$. If $\mathcal{D}[i]$ appears among the conjunctions of X then $V_X[i] = 1$. Otherwise it is 0.

Analogously, the vector representation of a conjunctive query q is a binary vector V_q of size N. The i-th element of the vector corresponds to pair $\mathcal{D}[i]$. If $\mathcal{D}[i]$ is one of the conjuncts of q, then $V_q[i] = 1$; otherwise it is 0.

Now we can define the similarity between context X and selection query q using their vector representations V_X and V_q as follows:

$$\text{sim}(q, X) = \cos(V_q, V_X) = \frac{V_q \cdot V_X}{|V_q||V_X|}.$$

We can additionally define the similarity between a query q and a set of contexts $\mathcal{X}$ as follows:

$$\text{sim}(q, \mathcal{X}) = \sum_{X\in\mathcal{X}} \text{sim}(q, X).$$

After this, the RANK SELECTION PROBLEM is fully defined. From the results of [12], it is easy to see the following:

THEOREM 1: *The* RANK SELECTION PROBLEM *is NP-hard.*

3.1 Approach

Given that Problem 1 is NP-hard, we have to think of approximation algorithms for solving it. Any such algorithm would require at least going through all preferences for each pair of tuples. This would have running time quadratic to the number of tuples in the database, not to mention the fact that the number of preferences themselves can be exponential. This time complexity is unacceptable for query-time operations on large databases. We therefore propose a hybrid approach that consists of some preprocessing, performed offline, followed by online query processing. The results of the offline computations make online processing of queries fast. The offline phase consists of two steps:

Step 1: For every class of preferences, construct a single order of the n input tuples. That is, for each preference class P_{X_i} find a permutation τ_i of the tuples in r such that

$$\tau_i = \arg\max_{\tau_i'} \textsc{Agree}(\tau_i', P_{X_i}). \tag{2}$$

The output of this step is a set of m ⟨context,order⟩ pairs of the form $\langle X_i, \tau_i\rangle$, where X_i is the context for the preferences in P_{X_i} and τ_i is the ordering of the tuples in r that (approximately) satisfies Equation 2. Due to the output ordering of tuples, each tuple t has a score that is associated with the position of t in each order τ_i. Thus, the score of tuple t in τ_i that corresponds to context X_i is:

$$s(t \mid X_i) = n - \tau_i(t) + 1,$$

where $\tau_i(t)$ represents the position of tuple t in order τ_i. Tuples in the head of the order have high scores, while low scores are reserved for tuples that are towards the tail of the order.

Step 2: Reduce the number of ⟨context,order⟩ pairs to be kept. That is, given the set of m initial pairs $\langle X_i, \tau_i\rangle$, find ℓ representative permutations $\bar{\tau}_1, \ldots, \bar{\tau}_\ell$ with $\ell < m$. These permutations partition the space of the m initial ⟨context,order⟩ pairs into ℓ groups. Each group i is characterized by order $\bar{\tau}_i$ and a disjunction of contexts $\bar{X}_i \subseteq \{X_1, \ldots, X_m\}$ such that for each $X_j \in \bar{X}_i$ order $\bar{\tau}_i$ is a *good representative* for the initial order τ_j. The score of tuple t in $\langle \bar{X}_i, \bar{\tau}_i\rangle$ is now given by:

$$s(t \mid \bar{X}_i) = n - \bar{\tau}_i(t) + 1. \tag{3}$$

The notion of a "good representation" of one order by another will become clear momentarily. For now, one can translate it as "closeness" or "similarity".

Given the offline computations, the query-time processing becomes computationally inexpensive. The only online task is to appropriately combine the a-priori formed rankings of the tuples to return a ranked answer to the given query.

Thus, instead of solving Problem 1 directly, we will solve the subproblems discussed below.

3.1.1 *The* ORDERING *problem*

PROBLEM 2 (ORDERING - OFFLINE): Given a relation $r = \{t_1, \ldots, t_n\}$ and a set of preferences P (all from the same class), find a permutation τ of the asserted tuples in r such that:

$$\tau = \arg\max_{\tau'} \text{AGREE}(\tau', P).$$

The ORDERING problem construes step 1. We discuss the complexity of the problem and the algorithmic solutions in Section 4.

3.1.2 *The* CLUSTERORDERS *problem*

We call the problem that step 2 defines CLUSTERORDERS problem. In order to quantify how well a permutation τ of the tuples in r is represented by another permutation ρ we need to define a distance measure between permutations over the same set of tuples. Several distance functions have been proposed in the literature for this purpose.We focus on two most well-known ones: the *Spearman footrule* and the *Kendall tau* [13].

- Given two permutations ρ and τ the *Spearman footrule* distance d_F is defined as the sum over all n tuples of the differences between the positions of those tuples in the two permutations:

$$d_F(\rho, \tau) \;=\; \sum_{i=1}^{n} |\tau(t_i) - \rho(t_i)|.$$

- The *Kendall tau* distance d_K counts the number of pairwise disagreements between the two permutations:

$$\begin{aligned} d_K(\rho, \tau) \;=\; & |\{(i,j) : i < j \text{ and } \rho(t_i) < \rho(t_j) \\ & \text{and } \tau(t_i) > \tau(t_j)\}|. \end{aligned}$$

We will use the generic symbol d to denote either of the two distance functions. Given the above definitions, the CLUSTERORDERS problem is defined as follows:

PROBLEM 3 (CLUSTERORDERS - OFFLINE): Assume an input consisting of m context-permutation pairs $\langle X_i, \tau_i \rangle$ and let T_m be the set of the m permutations over the tuples of relation r: $T_m = \{\tau_1, \ldots, \tau_m\}$. Find a set of $\ell < m$ permutations $T_\ell = \{\bar{\tau}_1, \ldots, \bar{\tau}_\ell\}$ such that

$$\text{cost}(T_\ell) \;=\; \sum_{\tau \in T_m} d(\tau, T_\ell) \qquad (4)$$

is minimized. The distance of a single permutation τ from a set of permutations T is defined as:

$$d(\tau, T) \;=\; \min_{\rho \in T} d(\tau, \rho).$$

We call the permutations in set T_ℓ *representative* permutations and associate with each representative permutation $\bar{\tau}_i$ a set of contexts $\bar{X}_j = \{X_i \mid \bar{\tau}_j = \arg\min_{j'} d(\tau_i, \bar{\tau_{j'}})\}$.

This problem exhibits some resemblance to the catalog segmentation problem proposed in [26]. However, there are significant differences between the two problems as well. For example, we are considering "ordered" catalogs, while they view catalogs as unordered sets. We discuss the complexity of the problem and algorithmic solutions in Section 5. Note that the problem of clustering orders into few clusters has also been studied in [24]. However, we adopt a different algorithmic approach and we give approximation bounds for our methods.

3.1.3 *The* QUERYING *problem*

With the ℓ representative orders at hand and the corresponding ℓ sets of contexts, we are ready to define the online query-processing problem:

PROBLEM 4 (QUERYING - ONLINE): For a given selection query q over relation r compute, using the output of step 2, the set $q_k(r) \subseteq q(r) \subseteq r$ with $|q_k(r)| = k$, such that $\forall t \in q_k(r)$ and $t' \in \{r - q_k(r)\}$ it holds that $\text{score}(t, q) > \text{score}(t', q)$, with $\text{score}(t, q) = \sum_{\bar{X}_i} \text{sim}(q, \bar{X}_i) \cdot s(t \mid \bar{X}_i)$.

Section 6 discusses the solution to this problem.

For the purposes of this paper, we will assume that the orderings are computed periodically as the set of preferences evolve and the database is updated. Note that on the web also, page ranks are not immediately readjusted as soon as a page is updated or deleted.

4. CONSTRUCTING ORDERS FROM PREFERENCES

We first discuss the complexity of Problem 2 and then describe three algorithms for solving it.

Consider the following MAXIMUM ACYCLIC SUBGRAPH problem, which is known to be NP-Hard [34]: for an input directed weighted graph G find the maximum-weight subgraph of G that is acyclic.

OBSERVATION 1: The ORDERING problem is as hard as the MAXIMUM ACYCLIC SUBGRAPH problem.

The connection between the MAXIMUM ACYCLIC SUBGRAPH and the ORDERING problem becomes intuitively clear via the X-preference graph. In the ORDERING problem the goal is for a given X-preference graph $G_X(r, E_X)$ to find a permutation τ of $r = \{t_1, t_2, \ldots, t_n\}$ that induces an acyclic subgraph $G_\tau(V, E_\tau)$ of G_X, such that the sum of the weights of the edges in E_τ is maximized. However this is exactly the MAXIMUM ACYCLIC SUBGRAPH problem on the preference graph G_X. A consequence of the above observation is the following corollary:

COROLLARY 1: The ORDERING problem is MAX-SNP complete.

The implication is that the ORDERING problem can be approximated within a fixed error ratio. However no PTAS can be found for this problem.

4.1 Algorithms

We next give three algorithms for the ORDERING problem. Two of them have a bounded approximation ratio, while the third is a heuristic that seems to perform extremely well in practice.

4.1.1 *The* PICK-PERM *Algorithm*

The PICK-PERM algorithm is inspired by the 2-approximation algorithm for the MAXIMUM ACYCLIC SUBGRAPH problem, that works as follows [5]: for a given input directed graph $G(V, E)$ with $|V| = n$ impose an arbitrary numbering of the nodes in V assigning randomly to each node a unique number from 1 to n. The set of edges E are then partitioned into two groups: $E_1 = \{e(i \to j) \in E \mid i > j\}$ and $E_2\{e(i \to j) \in E \mid i < j\}$, where i and j refer to the numbers assigned to the nodes in the imposed numbering. Clearly, the subgraphs G_1 and G_2 induced by E_1 and E_2 are acyclic. The algorithm outputs the subgraph having the maximum sum of edge weights.

In a similar spirit, PICK-PERM first constructs a random permutation of the tuples. Denote this permutation by τ. The reverse of permutation τ, τ^{-1} is also constructed. That is, if $\tau(t)$ is the position of tuple t in τ then $\tau^{-1}(t) = n - \tau(t) + 1$. The number of agreements between τ and τ^{-1} with the input context preferences is computed. The algorithm outputs either τ or τ^{-1}, favoring the one that has the larger number of agreements.

Algorithm 1 The PICK-PERM algorithm

```
Input: Relation $r = \{t_1, t_2, \ldots, t_n\}$; a set of preferences from a
    single class $P_X$ defined by context $X$.
Ouput: A pair $\langle X, \tau \rangle$ where $\tau$ is an ordering of the tuples in $r$
    such that as many of the preferences in $P_X$ are satisfied.
1: $\tau$ = random permutation of r
2: $\tau^{-1}$ = reverse($\tau$)
3: Return $\tau^* = \arg\max_{\{\tau,\tau^{-1}\}}\{\textsc{Agree}(\tau, P_X), \textsc{Agree}(\tau^{-1}, P_X)\}$
```

The following lemma shows that the PICK-PERM algorithm is a 2-approximation algorithm for the ORDERING problem.

LEMMA 1: *Let P_X be the set of input preferences with context X. Denote by A_{PP} be the number of agreements between the order output by the* PICK-PERM *algorithm and the input set of preferences and let N_{opt} be number of agreements between the optimal permutation of the tuples with the input preferences. Then, $A_{PP} \geq N_{opt}/2$.*

PROOF. First observe that:

$$N_{\text{opt}} \leq \textsc{Agree}(\tau, P_X) + \textsc{Agree}(\tau^{-1}, P_X).$$

The solution given by the PICK-PERM has the following property:

$$\begin{aligned} A_{\text{PP}} &= \max_{\{\tau,\tau^{-1}\}} \textsc{Agree}(\tau, P_X) + \textsc{Agree}(\tau^{-1}, P_X) \\ &\geq (\textsc{Agree}(\tau, P_X) + \textsc{Agree}(\tau^{-1}, P_X))/2 \\ &\geq N_{\text{opt}}/2. \end{aligned}$$

□

4.1.2 *The* GREEDY-ORDER *Algorithm*

The GREEDY-ORDER (Algorithm 2) is an adaptation of the algorithm proposed in [12]. It is also a 2-approximation algorithm for the ORDERING problem. The intuition behind the algorithm is straightforward, particularly if we consider it operating on the X-preference graph. At every step (*rank*) a greedy selection is made.

Algorithm 2 The GREEDY-ORDER algorithm

```
Input: Relation $r = \{t_1, t_2, \ldots, t_n\}$; a set of preferences from a
    single class $P_X$.
Ouput: A pair $\langle X, \tau \rangle$ where $\tau$ is an ordering of the tuples in $r$
    such that as many of the preferences in $C_X$ are satisfied.
1: CANDIDATES = $\{t_1, t_2, \ldots, t_n\}$
2: rank = 0
3: for all $i \in \{1, \ldots, n\}$ do
4:     $p(t_i) = \sum_{j=1}^{n} w_X(t_i \to t_j) - \sum_{j=1}^{n} w_X(t_j \to t_i)$
5: end for
6: while CANDIDATES $\neq \emptyset$ do
7:     rank = rank + 1
8:     $t_v = \arg\max_{t_u \in \textsc{Candidates}} p(t_u)$
9:     $\tau(t_v)$ = rank
10:    CANDIDATES = CANDIDATES $- \{t_v\}$
11:    for all $t \in$ CANDIDATES do
12:        $p(t) = p(t) - w_X(t \to t_v) + w_X(t_v \to t)$
13:    end for
14: end while
```

The tuple t_v (node in the X-preference graph) with the highest $p(t_v)$ value is picked and it is assigned in position *rank* in the output order. The $p(t_v)$ value of a tuple is its out-degree minus its in-degree in the preference graph that is induced after every greedy step.

4.1.3 *The* MC-ORDER *Algorithm*

Let $G_X(V_X, E_X)$ be the X-preference graph, where as before V_X corresponds to the set of tuples of r. The MC-ORDER algorithm starts by creating the graph $\bar{G}_X(V_X, \bar{E}_X)$, where $\bar{E}_X$ are all edges in E_X but reversed. A random walk is performed on graph $\bar{G}_X$ and the nodes V_X are ranked according to their values in the stationary distribution of this random walk.

Recall that a directed edge $e(t_i \to t_j) \in E_X$ indicates that there exists a preference in P_X for which tuple t_i is preferred over tuple t_j. The intuitive meaning of the reversed edge $\bar{e}(t_j \to t_i)$ in the graph $\bar{G}_X$ is that since t_i is preferred to t_j, in the random walk t_j propagates some of its weight to t_i.

Let $\bar{A}$ and A be the adjacency matrices of graphs $\bar{G}_X$ and G_X respectively. Then it holds that $\bar{A}[i, j] = A[j, i] = w_X(j \to i)$ (as in 1). Now consider the Markov chain on the following stochastic matrix:

$$M = \alpha S(\bar{A}) + (1 - \alpha)E.$$

The operator S converts the input matrix $\bar{A}$ into a stochastic matrix. The matrix E is a matrix with $E[u, v] = \frac{1}{n}$ for all (t_u, t_v) pairs. Finally, parameter α is a real number between 0 and 1. This parameter allows the random walk to make random jumps to arbitrary nodes at any time and guarantees the convergence of the random walk to a stationary probability distribution. We denote this probability distribution by π. The algorithm computes an ordering τ of the nodes of the graph, such that $\tau(t_u) < \tau(t_v)$ iff $\pi(t_u) > \pi(t_v)$. Ties are broken arbitrarily.

Although no approximation guarantees are known for the algorithm, the intuition behind it is simple. One naturally expects that the random walk would end-up more often in a node that is preferred more. This means that the value of such a node in the stationary distribution would be higher and this node will be placed higher in the output order τ. As we shall see, this algorithm gives high-quality results in practice.

5. FINDING REPRESENTATIVE ORDERS

We first discuss the complexity of the CLUSTERORDERS problem and then present algorithms for dealing with it. First we observe the following theorem:

THEOREM 2: *The* CLUSTERORDERS *problem is NP-hard for both the Kendall tau and the Spearman footrule distance metrics.*

Note that the special case of the CLUSTERORDERS problem where $\ell = 1$ and the cost function employs distance d_F can be solved in polynomial time using a standard algorithm for maximal bipartite graph matching [14]. This reveals an artifact of the segmented versions of standard optimization problems (see [26] for a thorough discussion). While the simple version of the problem ($\ell = 1$ and d being d_F) is polynomially solvable, the segmented version of the problem $\ell > 1$ is NP-hard.

5.1 Algorithms

We next present two algorithms for the CLUSTERORDERS problem. The input to both of them is a set of m pairs of the form $\langle X_i, \tau_i \rangle$ with $1 \leq i \leq m$. Each X_i is a context and each τ_i is a permutation of the n tuples of relation r. The output is a set of ℓ pairs of the form $\langle \bar{X}_i, \bar{\tau_i} \rangle$. Each $\bar{X}_i$ is a set of contexts ($\bar{X}_i \subseteq \{X_1, \ldots, X_m\}$) such that $\forall X_j \in \bar{X}_i$, it holds that $\bar{\tau_i} = \arg\min_{i'} d(\bar{\tau_{i'}}, \tau_j)$. Additionally, each initial context X_j is mapped to a unique final set of contexts $\bar{X}_i$.

Before discussing the algorithms for the CLUSTERORDERS problem we provide some useful results that will help with the analysis of the algorithms.

FACT 1: (due to [13]) For any two permutations ρ and τ over the same set of n objects, the following inequality is true:

$$d_K(\tau, \rho) \leq d_F(\tau, \rho) \leq 2 d_K(\tau, \rho).$$

The consequence of this fact is that the optimal solution to the CLUSTERORDERS problem for distance d being d_F is a 2-approximation of the optimal solution for the CLUSTERORDERS problem and distance being d_K. Thus, any bounded factor approximation algorithm for CLUSTERORDERS using d_F, is also a bounded factor approximation algorithm for the same problem using d_K.

FACT 2: Triangle inequality holds for both d_F and d_K. That is, if τ_1, τ_2 and τ_3 are any three permutations over a set of n objects. Then, the following inequalities are true:

$$d_F(\tau_1, \tau_2) + d_F(\tau_2, \tau_3) \quad \leq \quad d_F(\tau_1, \tau_3).$$

and

$$d_K(\tau_1, \tau_2) + d_K(\tau_2, \tau_3) \quad \leq \quad d_K(\tau_1, \tau_3).$$

Now consider a restricted version of the CLUSTERORDERS problem, where the ℓ representatives are constrained to be chosen from the set of initial input orders. We call this restricted version of the problem the DISCRETE-CLUSTERORDERS problem. The following adaptation of the folklore theorem for clusters of points is true:

THEOREM 3: *Let* OPT *be the cost of the optimal solution of the* CLUSTERORDERS *problem and* D-OPT *the cost of the optimal solution for the* DISCRETE-CLUSTERORDERS *problem, for distance metric d. Then it holds that:*

$$\text{D-OPT} \leq 2\text{OPT}.$$

5.1.1 *The* GREEDY *algorithm*

Algorithm 3 The GREEDY algorithm for finding representative orders

Input: A set of m orders of the n tuples in r: $T_m = \{\tau_1, \ldots, \tau_m\}$.
Ouput: A set of representative ℓ orders $T_\ell = \{\bar{\tau}_1, \ldots, \bar{\tau}_\ell\}$.

```
1: CANDIDATES = $T_m$
2: for all $i = m$ down to $k$ do
3:     $\rho = \arg\min_{i \in \text{CANDIDATES}} \sum_{\tau \in T_m} d(\tau, \text{CANDIDATES} - \{\tau_i\}) - \sum_{\tau \in T_m} d(\tau, \text{CANDIDATES})$
4:     CANDIDATES = CANDIDATES $- \{\rho\}$
5: end for
6: $T_\ell \leftarrow$ CANDIDATES
```

The GREEDY algorithm (Algorithm 3) for the CLUSTERORDERS problem is a variation of a greedy algorithm provided in [11] for the *facility-location problem*. The apparent similarity between the two problems is that the ℓ output orders can be considered as the facilities that serve the initial m input orders ($\ell < m$). The GREEDY algorithm starts by considering all the m input orders as candidates to be representatives. In each iteration the algorithm removes from the candidate set the order which when removed, causes the least increase in the total cost. The algorithm stops when ℓ orders remain in the candidate set.

PROPOSITION 1: (due to [11]) The GREEDY algorithm is an $O(\log m)$ approximation algorithm for the DISCRETE-CLUSTERORDERS problem with m initial input orders. Therefore if cost(G) is the cost of the solution found by the GREEDY algorithm and D-OPT the cost of the optimal solution for the DISCRETE-CLUSTERORDERS problem, then it holds that:

$$\text{cost(G)} \quad \leq \quad O(\log m)\text{D-OPT}.$$

The above proposition holds irrespective of whether the cost is calculated using d_F or d_K, due to Fact 2.

PROPOSITION 2: The GREEDY algorithm is an $O(\log m)$ approximation algorithm for the CLUSTERORDERS problem with m initial input orders. That is, if cost(G) is the cost of the solution found by the GREEDY algorithm and OPT the cost of the optimal solution for the CLUSTERORDERS problem, then it holds that:

$$\text{cost(G)} \quad \leq \quad O(\log m)\text{OPT}.$$

PROOF. Follows from Proposition 1 and Theorem 3.

5.1.2 *The* FURTHEST *algorithm*

The FURTHEST algorithm (Algorithm 4) is a top-down algorithm. It is inspired by the *furthest-first traversal* algorithm for which Hochbaum and Shmoys [22] showed that it achieves a 2-approximation for the p-centers clustering problem.

The algorithm starts by picking an arbitrary order as a representative, say order τ_{i_1}, and adds it to the output orders $T_\ell = \{\tau_{i_1}\}$, while it removes it from T_m. Then it picks order τ_{i_2}, which is furthest from τ_{i_1}. The algorithm continues for ℓ steps. Finally, it assigns the remaining orders in T_m to their closest representative.

Algorithm 4 The FURTHEST algorithm for finding representative orders

```
Input: A set of $m$ orders of the $n$ tuples in $r$: $T_m = \{\tau_1, \tau_2, \dots, \tau_m\}$
Ouput: A set of representative $\ell$ orders $T_\ell = \{\bar{\tau}_1, \bar{\tau}_2, \dots, \bar{\tau}_\ell\}$
 1: $T_\ell = \emptyset$
 2: Pick arbitrary $\bar{\tau} \in T_m$
 3: $T_\ell = T_\ell \cup \{\bar{\tau}\}$
 4: $T_m = T_m - \{\bar{\tau}\}$
 5: for all $i = 2$ to $\ell$ do
 6:     $\bar{\tau} = \arg\max_{\tau'} d(\tau', T_\ell)$
 7:     $T_\ell = T_\ell \cup \{\bar{\tau}\}$
 8:     $T_m = T_m - \{\bar{\tau}\}$
 9: end for
10: for all $\tau \in T_m$ do
11:     Assign $\tau$ to its closest order in $T_\ell$
12: end for
```

5.2 Refinements

Both the above algorithms output ℓ representative orders from the set of initial input orders. That is, in both cases $T_\ell \subseteq T_m$. Assume that the m initial orders are partitioned into ℓ groups and let this partitioning be $\mathcal{G} = \{\mathcal{G}_1, \dots, \mathcal{G}_\ell\}$. Consider partition $\mathcal{G}_i$ of the input orders that has as a representative order $\bar{\tau}_i$. The above algorithms do not guarantee that $\bar{\tau}_i = \arg\min_{\tau \in \mathcal{G}_i} \sum_{\tau' \in \mathcal{G}_i} d(\tau, \tau')$. To overcome this shortcoming we introduce two possible refinements:

1. **Discrete Refinement:** For each partition $\mathcal{G}_i$ of the input orders, replace the representative $\bar{\tau}_i$ found by the GREEDY or the FURTHEST algorithm with $\bar{\tau}_i{}^* \in \mathcal{G}_i$ such that
$$\bar{\tau}_i{}^* = \arg\min_{\tau \in \mathcal{G}_i} \sum_{\tau' \in \mathcal{G}_i} d(\tau, \tau').$$

2. **Continuous Refinement:** For each partition $\mathcal{G}_i$ of the input orders, replace the representative $\bar{\tau}_i$ found by the GREEDY or the FURTHEST algorithm with $\bar{\tau}_i{}^*$ such that
$$\bar{\tau}_i{}^* = \arg\min_{\tau} \sum_{\tau' \in \mathcal{G}_i} d(\tau, \tau').$$
That is pick *any* permutation from the set of all possible permutations that best represents the orders in partition $\mathcal{G}_i$. The problem of picking such a representative when d is d_K is computationally hard. Algorithms from [14] and [15] can be used for approximating the best representative.

Neither of the refinements changes the partitioning of the m initial orders; only the representative of each group of the partition changes so that the total cost of the partitioning is reduced.

6. RANKED TOP-K QUERIES

This section describes the solution of Problem 4: how the computations made in the offline steps are exploited to provide ranked top-k results. The setting is the following: There are ℓ different orderings of all the tuples of relation r. Each ordering $\bar{\tau}_i$ is associated with a set of contexts $\bar{X}_i \subseteq \{X_1, \dots, X_m\}$ forming ℓ pairs $\langle \bar{X}_i, \bar{\tau}_i \rangle$. Additionally, each tuple $t \in r$ in each such pair i is associated with the $s(t \mid \bar{X}_i)$ as defined in Equation 3.

We adapt *TA algorithm* [18] to retrieve the top-k answers to a query. Another alternative would be to use the *MedRank* algorithm proposed in [17]. In the TA algorithm there are two modes of access to the data. *Sorted access* obtains the score of an element in an order by traversing the order of the tuples sequentially from the top. *Random access* obtains the score of a tuple in an order in one access. The algorithm in our setting works as follows:

1. Do a round-robin access to each one of the ℓ orderings of the tuples. As a tuple t with $t \in q(r)$ is seen in some ordering $\bar{\tau}_i$, find the score $s(t \mid \bar{X}_j)$ with $j \neq i$ by finding the position of t in all $\bar{\tau}_j$'s. The final score of tuple t for the query q is computed as:
$$\text{score}(t, q) = \sum_{\bar{X}_i} \text{sim}(q, \bar{X}_i) \cdot s(t \mid \bar{X}_i).$$

2. Not all tuples of each sorted list are accessed in step 1. Let $s(\underline{t_j} \mid \bar{X}_i)$ be the score of the last visited tuple of ordering $\bar{\tau}_i$ by the end of the j-th round-robin cycle. Then, the threshold value **s** is defined to be $\mathbf{s} = \sum_{\bar{X}_i} \text{sim}(q, \bar{X}_i) \cdot s(\underline{t_j} | \bar{X}_i)$. The algorithm halts when k tuples with score values greater or equal to **s** have been seen.

3. Among all the tuples seen, output those k with the highest value for $\text{score}(t, q)$.

Note that by the end of step 2, for any tuple t' that has not yet been seen in the round-robin access of the data, it holds that $\text{score}(t', q) \leq$ **s**. The complexity of the algorithm is linear in the number of input orderings. Consider relation r with schema $R(A_1, \dots, A_d)$. Let the cardinality of the active domain of each attribute $\text{Dom}(A_i)$ be $|\text{Dom}(A_i)|$. There can be $2^{|\text{Dom}(A_1)| + \dots + |\text{Dom}(A_d)|}$ different contexts. If a different order had been generated for each one of them, the TA algorithm would need $O(2^{|\text{Dom}(A_1)| + \dots + |\text{Dom}(A_d)|})$ time. The reduction of the number of orders to ℓ representative ones leads to a reduction of the running time to $O(\ell)$.

7. HANDLING INDIFFERENT TUPLES

In this section, we discuss the implications of the existence of indifferent tuples on the algorithms we proposed in the previous sections. It is natural to assume that an indifferent tuple is less important than *any* asserted tuple. This is because the preferences expressed by users are *supportive*. Therefore, if users have not bothered to compare a specific tuple with any other tuple in the dataset, then this tuple is a rather unimportant one. Secondly, when it comes to comparing two indifferent tuples with each other, we cannot conclude, in the absence of any evidence, that one of them is more or less important than the other.

Having the above in mind, one could deal with the ORDERING problem by completely ignoring the indifferent tuples and just considering the asserted ones. In that case, all the proposed algorithms in Section 4 provide orderings of the asserted tuples. Once such an order is constructed, the indifferent tuples are assumed to follow the asserted tuples in the order. We thus obtain a total ordering of the asserted tuples, followed by a "bucket" of indifferent tuples, with no explicit ordering between them. Let us call such an ordering a *mixed* order.

We now need to modify the CLUSTERORDERS problem to deal with mixed orders instead of total orders. For this, the definition of distance functions between mixed orders is necessary. Thus, we modify the definitions of Spearman footrule and Kendall tau to operate on the mixed orders. Assume a mixed order ρ on n tuples and let n_ρ be the number of totally ordered tuples and $n - n_\rho$ the number of indifferent tuples associated with it. For each such mixed order ρ we denote by o_ρ the total order on its n_ρ asserted tuples and by b_ρ the corresponding bucket of indifferent tuples.

The modifications are straightforward given the following fact: For a mixed order ρ and tuple t we assume that:

$$\rho(t) = \begin{cases} o_\rho(t) & \text{if } t \text{ is asserted,} \\ n_\rho + \frac{n-n_\rho}{2} & \text{if } t \text{ is indifferent.} \end{cases}$$

In the above, $o_\rho(t)$ denotes the position of the asserted tuple t in order o_ρ. The positions of the indifferent tuples are all the same. Though the modified Spearman footrule remains a metric (Fact 2 still holds), this is not the case for the modification of Kendall tau. That is, the approximation guarantees we have provided for the GREEDY algorithm do not immediately hold, when dealing with mixed orders and Kendall distances. However, this obstacle might be overcome by using the result from [16] that puts the modified d_K and d_F functions in the same equivalence class. Thus a bounded-factor approximation algorithm for the CLUSTERORDERS problem using d_F is also a bounded-factor approximation algorithm for d_K.

Finally, the TA algorithm used in the online QUERYING problem can also operate on mixed orders. For this, we assume a random ordering of the tuples in all the buckets of indifferent tuples. Consider again the mixed order ρ. In this case the score of a tuple t in ρ is defined to be: $s(t \mid \rho) = n - \rho(t) + 1$, with $\rho(t)$ as defined above.

8. EXPERIMENTAL RESULTS

8.1 Experiments on synthetic datasets

Experiment I

In the first experiment we empirically evaluate the algorithms for the ORDERING problem. To that aim, we generate two types of preferences: (a) *general acyclic* and (c) *strictly acyclic*. The use of acyclic preferences allows us to have a known ground truth for comparing the performance of the algorithms.

For general acyclic preferences, the preference graph is generated as follows. First we fix an order on the tuples and then we generate preferences that respect this order. That is, we generate a directed edge from i to j with $\frac{1}{2} \leq w(i \to j) \leq 1$ only if i precedes j in the fixed order. This process results in the generation of the corresponding backward edge with weight $w(j \to i) = 1 - w(i \to j) < \frac{1}{2}$. The parameter $p_c \in [0, 1]$ is used to specify the probability that a preference has been generated for a pair of tuples. Note that weight $w(i \to j)$ corresponds to the value of effective preference EFF-P(i, j, X). Note that for this experiment we assume a single context, say X.

The generation process for the strictly acyclic case is similar. The only difference is that the weight associated with each forward edge is set to 1 (and 0 for the corresponding backward edge).

In the datasets so generated, we know that the correct underlying order of the tuples is the one that was used for generating the preferences. Thus, we can compare the performance of the three algorithms with the performance of an optimal algorithm OPT, that outputs the order used for the data generation. Figure 2, shows the ratio of $\frac{\text{AGREE}(A)}{\text{AGREE}(\text{OPT})}$, for $A = \{$PICK-PERM, MC-ORDER, GREEDY-ORDER$\}$, as a function of the value of p_c used for the generation process. The value of the ratio has been averaged over sufficient number of runs to obtain tight confidence intervals. The results are insensitive to the number of tuples ($n \geq 1000$).

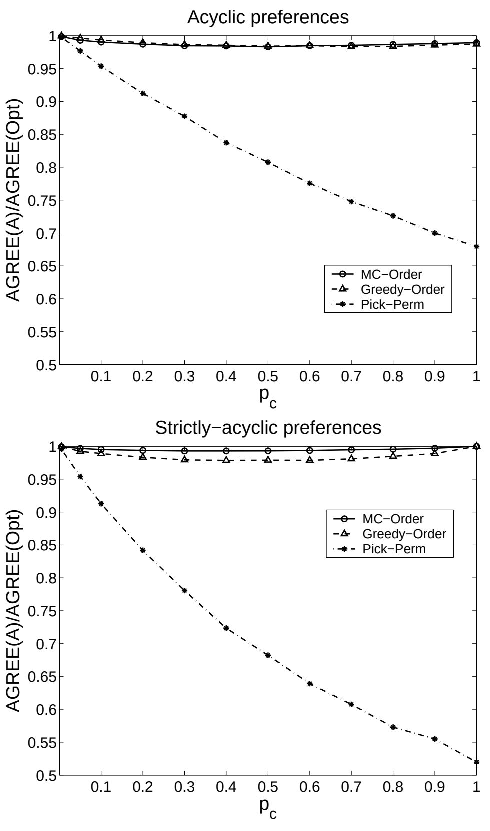

Figure 2: Algorithms' performance for the ORDERING problem

In both the acyclic and the strictly acyclic case, the three algorithms have performance identical to the performance of OPT for small values of p_c. This is due to the fact that many orders, from the universe of all possible orders, are close to optimal when no preference has been specified for most of the pairs of tuples. As the value of p_c increases, the differences in the algorithms' performance become more pronounced. The PICK-PERM deviates from the optimal reaching a ratio close to its lower bound, while MC-ORDER and GREEDY-ORDER maintain a ratio close to 1.0. Notice that in the strictly-acyclic case MC-ORDER is slightly better than GREEDY-ORDER. The reason for this is that in such preference graphs the MC-ORDER algorithm cannot err as it transfers weights in the right direction (from tuples that are in the tail of the order to tuples that are in the earlier positions).

Experiment II

The second experiment aims at testing the quality of the algorithms for the CLUSTERORDERS problem. For this experiment we assume there are no indifferent tuples. We generate synthetic datasets with the following procedure. Every dataset is characterized by 5 parameters: n, m, ℓ, *noise* and *type*. Here n is the number of tuples in

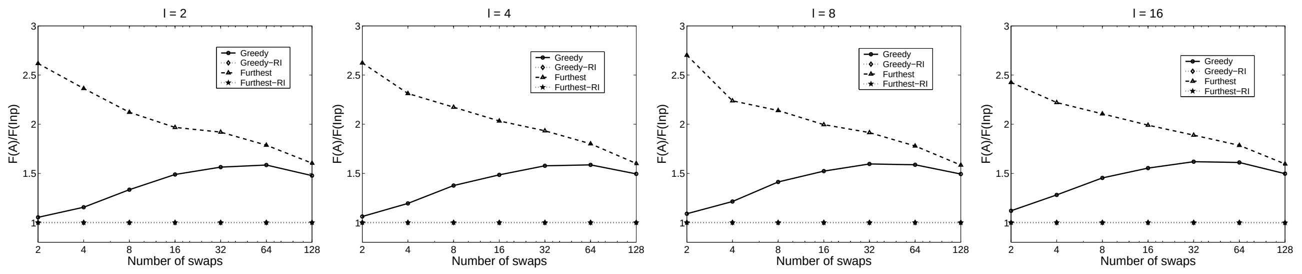

Figure 3: Algorithms' performance for the ClusterOrders problem, (noise type = swaps)

.

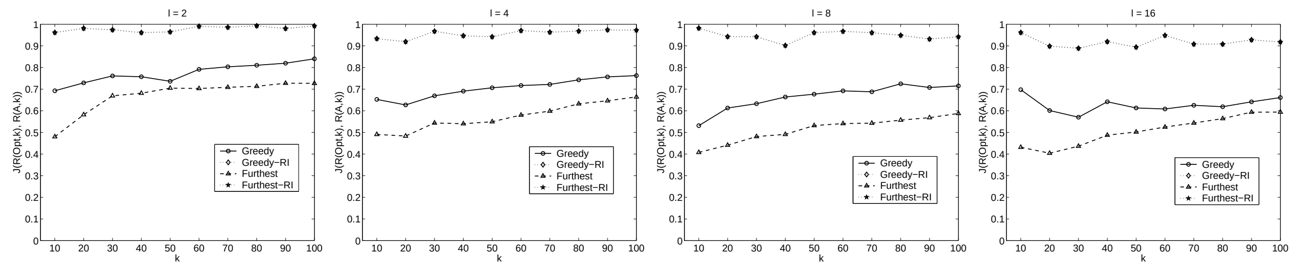

Figure 4: Jaccard coefficient of the top-k results.

each of the orders, m the number of input orders, and ℓ the number of true underlying clusters. Initially, we generate ℓ random orders by sampling uniformly at random the space of all possible permutations of n elements. These initial orders form the centers around which we build each one of the clusters. The task of the algorithms is to rediscover the clustering model used for the data generation. Given a cluster center, each order from the same cluster is generated by adding to the center a specified amount of *noise* of a specific *type*. We consider two types of noise: *swaps* and *shifts*. For realizing a swap two random tuples from the initial order are picked and their positionings in the ordering are exchanged. For the shifts we pick a random tuple and we move it to a new position, either earlier (or later) in the order. All tuples that are between the new and the old positions of the tuple are shifted one position down (or up). The amount of noise is the number of swaps or shifts we make.

We experiment with datasets generated for the following parameters: $n = 500$, $m = 1000$, $\ell = \{2, 4, 8, 16\}$, noise $= \{2, 4, 8 \ldots, 128\}$ and for both swaps and shifts. Figure 3 shows the performance of the algorithms as a function of the amount of noise. We only show the results for swaps since the results for shifts exhibited similar trends. The y axis is the ratio: $\frac{\mathrm{F}(A)}{\mathrm{F}(\textsc{Inp})}$, for $A = \{\textsc{Greedy}, \textsc{Furthest}\}$. The $F(A)$ is the total cost of the solution provided by algorithm A when footrule distance is used as a distance measure between orders. The quantity $F(\textsc{Inp})$ corresponds to the cost of the clustering structure (Equation 4) used in the data-generation process. Note that $F(\textsc{Inp})$ need not be minimal, particularly in datasets with high values of noise. As large amount of noise may destroy the preordained clustering structure. Therefore, the above ratio can only be viewed as an indication of how well the methods work.

We have plotted the performance of the algorithms without and with refinements to their outputs. In the case of the latter the algorithms' names are suffixed with "RI" (e.g. Greedy-RI). For these plots we are considering refinement 1, (results for refinement 2 were similar and thus omitted).

We see that: the Greedy algorithm (that also has the approximation bound) performs steadily better than Furthest when no refinements are applied. As expected the performance of the Greedy algorithm degrades as the amount of noise increases. On the other hand, Furthest exhibits the opposite trend. This surprising behavior can be explained considering the way the latter algorithm proceeds. In the cases of low noise (small number of swaps) Furthest picks as cluster centers orders that are far apart. These orders may indeed belong to different clusters, but they are not necessarily the "true" cluster centers. The algorithm is prone to making such wrong judgments for the following reason: due to the low noise, the distance between any two orders belonging to different clusters is more or less identical and large. Thus, the algorithm cannot distinguish between the real cluster center and any order that comes from the same cluster. Therefore, it makes correct grouping of the input orders, but it is prone to picking the "wrong" representatives.

Finally,we note that the refinement step is very effective in improving the performance of both Greedy and Furthest. In fact, their performance becomes nearly identical to the input preordained clustering. This observation suggests that even without refinements both Greedy and Furthest are able to do the correct groupings, though they do not succeed in rediscovering the correct cluster centers as well.

Experiment III

The third experiment (Figure 4) tests the accuracy of the top-k results obtained using only the representative orders when compared with the top-k results obtained using all available orderings. That is, we want to test the loss in accuracy due to compression. We quantify this accuracy as follows: let $R(\textsc{Opt}, k)$ be the top-k tuples returned by the algorithm using all the available orderings, and $R(A, k)$ the corresponding top-k tuples returned when only

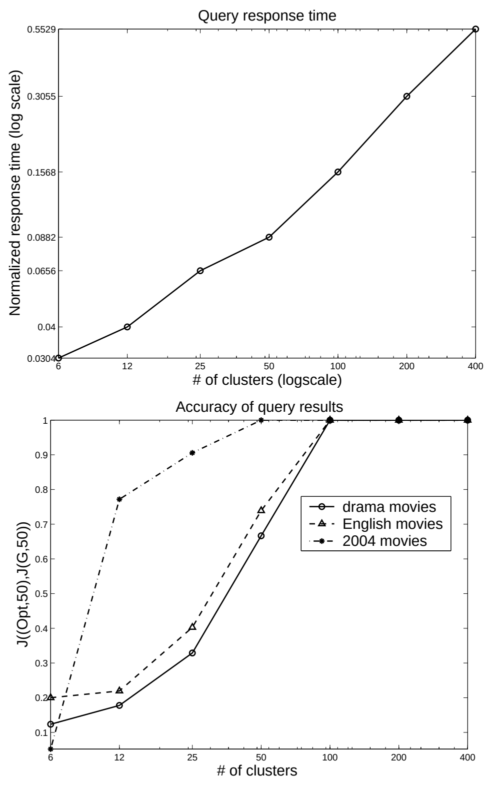

Figure 5: Accuracy/efficiency tradeoffs for IMDB dataset.

the representative orders, as output by algorithm A, are taken into account. We compare the two top-k answers as sets using the Jaccard Coefficient defined as follows:

$$J(R(\textsc{Opt}, k), R(A, k)) = \frac{|R(\textsc{Opt}, k) \cap R(A, k)|}{|R(\textsc{Opt}, k) \cup R(A, k)|}.$$

The Jaccard Coefficient takes real values between 0 and 1 and the higher its value the more similar the two sets of tuples are. Figure 4 shows the value of the coefficient for different values of k, when $A = \{\textsc{Greedy}, \textsc{Greedy} - \text{RI}, \textsc{Furthest}, \textsc{Furthest} - \text{RI}\}$. For this experiment the datasets were generated the same way as for experiment II. We show plots for $\ell = \{2, 4, 8, 16\}$, while in this case we fix the level of noise to 64 swaps. The values of n and m are fixed to 500 and 1000 respectively. The results were insensitive to the latter parameters.

We see that the refined results of both the algorithms are nearly identical and the accuracy is very high, even for small values of k. Even without refinement the values of the Jaccard Coefficients are large. Thus even when only small number of tuples is returned to the user, the information lost by looking at the clustered orders instead of all orders is minimal.

8.2 Real Dataset

Finally, we test our methodology on a real dataset. For this purpose we use the IMDB movie dataset (www.imdb.com). From the dataset we extract for each movie information related to its genre, language, production year, directors and actors. We include movies with language being English, French, German, Spanish, Japanese, Finnish or Swedish. We further select movies with ratings (information available in the dataset) greater than 7.0. This subset consists of nearly $32,000$ movies. From this data, we construct the following relational table: $r =$ (TID, title, genre, year, language, actor, director).

For the purpose of the experiment we automatically generate preferences via association-rules mining [1]. The rationale behind this automatic generation of preferences is the following. We say that $\{A_1 = a \succ A_2 = b \mid X\}$ if conf$(X \rightarrow a) >$ conf$(X \rightarrow b)$, where conf$(X \rightarrow a)$ is the confidence of the association rule $X \rightarrow a$ in the database, i.e., conf$(X \rightarrow a) = \frac{\text{fr}(X \wedge a)}{\text{fr}(X)}$. The underlying assumption is that when an attribute value a occurs together with context X more often than the attribute value b, then this implies that a is also preferred to b over X. Using a confidence level of 0.2 we obtained 883 different classes of preferences, which we used in our experiments. Note that in the real datasets indifferent tuples appear and therefore for this experiment we take into consideration their implications in the different algorithmic steps.

Figure 5 shows the query response time and accuracy tradeoff for different number of representative clusters. The query response times have been normalized using the time needed for number of clusters equal to 883 as the base. The accuracy has been measured in terms of the Jaccard Coefficient. The MC-Order was used for constructing orders from preferences and Greedy was used for clustering the constructed orders and finding the cluster representatives.

The results are very encouraging; as the figure shows the time requirements increase linearly as a function of the number of clusters. The absolute response time on our unoptimized implementation running on a 2.3 GHz NT workstation was less than 3 seconds for 50 clusters. More interestingly, although the association-rule mining phase has created 883 classes of preferences, clustering them in 50 clusters provided satisfactory levels of accuracy for most of the queries we tested.

9. CONCLUSIONS

We proposed a framework and algorithmic solution to the problem of abundance deluging the database-centric web applications. Our framework is based on taking advantage of the user preferences to precompute a few representative tuple orderings and use them to expeditiously provide ranked answers to database queries factoring in the information contained in the query. Our approach is similar in spirit to the successful strategy employed by web search engines to deal with the problem of abundance of web pages: precompute page rank and use it to order the response to search queries.

Our preference language is natural and intuitive and does not require users to specify numeric importance scores to tuples. Rather, users provide context-dependent choices (or they are mined), which are incorporated in the formulation of final ranking in a combinatorial manner. The language is set-oriented and allows a single preference to specify choices between a large number of pairs of tuples. Two tuples can be ordered differently depending on the context. To enable mass collaboration, we admit choices that can be contradictory and contain cycles, and reconcile them democratically.

We performed extensive experiments, using both synthetic and real data, to empirically study the performance characteristics of the proposed algorithms. These experiments show that our proposed solution achieves high accuracy even when only a small number of representative orderings are kept, saving storage and making responses to top-k ranked answers fast.

In the future, we would like to study incremental algorithms for maintaining orderings as the preferences and database evolve. We would also like to investigate how to make our solution spam-resistant in the presence of malicious users.

Acknowledgements

We would like to thank Heikki Mannila for his comments on an earlier version of this paper.

10. REFERENCES

[1] R. Agrawal, T. Imielinski, and A. N. Swami. Mining association rules between sets of items in large databases. In *SIGMOD*, pages 207–216, 1993.

[2] R. Agrawal and E. L. Wimmers. A framework for expressing and combining preferences. In *SIGMOD*, pages 297–306, 2000.

[3] S. Agrawal, S. Chaudhuri, G. Das, and A. Gionis. Automated ranking of database query results. In *CIDR*, 2003.

[4] A. Balmin, V. Hristidis, and Y. Papakonstantinou. Objectrank: Authority-based keyword search in databases. In *VLDB*, pages 564–575, 2004.

[5] B. Berger and P. W. Shor. Approximation algorithms for the maximum acyclic subgraph problem. In *SODA*, pages 236–243, 1990.

[6] G. Bhalotia, A. Hulgeri, C. Nakhe, S. Chakrabarti, and S. Sudarshan. Keyword searching and browsing in databases using BANKS. In *ICDE*, pages 431–440, 2002.

[7] S. Borzsonyi, D. Kossmann, and K. Stocker. The skyline operator. In *ICDE*, pages 421–430, 2001.

[8] S. Brin and L. Page. The anatomy of a large-scale hypertextual web search engine. *Computer Networks*, 30(1-7):107–117, 1998.

[9] S. Chaudhuri, G. Das, V. Hristidis, and G. Weikum. Probabilistic ranking of database query results. In *VLDB*, pages 888–899, 2004.

[10] J. Chomicki. Preference formulas in relational queries. *ACM Trans. Database Syst.*, 28(4):427–466, 2003.

[11] M. Chrobak, C. Kenyon, and N. E. Young. The reverse greedy algorithm for the metric k-median problem. In *COCOON*, 2005.

[12] W. W. Cohen, R. E. Schapire, and Y. Singer. Learning to order things. In *NIPS*, 1997.

[13] P. Diaconis and R. Graham. Spearman's footrule as a measure of disarray. *J. of the Royal Statistical Society*, 39(2):262–268, 1977.

[14] C. Dwork, R. Kumar, M. Naor, and D. Sivakumar. Rank aggregation methods for the web. In *WWW*, pages 613–622, 2001.

[15] R. Fagin, R. Kumar, M. Mahdian, D. Sivakumar, and E. Vee. Comparing and aggregating rankings with ties. In *PODS*, pages 47–58, 2004.

[16] R. Fagin, R. Kumar, and D. Sivakumar. Comparing top k lists. In *SODA*, pages 28–36, 2003.

[17] R. Fagin, R. Kumar, and D. Sivakumar. Efficient similarity search and classification via rank aggregation. In *SIGMOD*, pages 301–312, 2003.

[18] R. Fagin, A. Lotem, and M. Naor. Optimal aggregation algorithms for middleware. In *PODS*, pages 102–113, 2001.

[19] F. Geerts, H. Mannila, and E. Terzi. Relational link-based ranking. In *VLDB*, pages 552–563, 2004.

[20] L. Guo, F. Shao, C. Botev, and J. Shanmugasundaram. XRANK: Ranked keyword search over XML documents. In *SIGMOD*, pages 16–27, 2003.

[21] T. H. Haveliwala. Topic-sensitive pagerank. In *WWW*, pages 517–526, 2002.

[22] D. S. Hochbaum and D. B. Shmoys. A best possible heuristic for the k-center problem. *Mathematics of Operations Research*, pages 180–184, 1985.

[23] A. Huang, Q. Xue, and J. Yang. Tuplerank and implicit relationship discovery in relational databases. In *WAIM*, 2003.

[24] T. Kamishima and J. Fujiki. Clustering orders. In *Discovery Science*, pages 194–207, 2003.

[25] W. Kießling. Foundations of preferences in database systems. In *VLDB*, pages 311–322, 2002.

[26] J. Kleinberg, C. Papadimitriou, and P. Raghavan. Segmentation problems. In *STOC*, pages 473–482, 1998.

[27] G. Koutrika and Y. Ioannidis. Constrained optimalities in query personalization. In *SIGMOD*, pages 73–84, 2005.

[28] G. Koutrika and Y. E. Ioannidis. Personalization of queries in database systems. In *ICDE*, pages 597–608, 2004.

[29] G. Koutrika and Y. E. Ioannidis. Personalized queries under a generalized preference model. In *ICDE*, pages 841–852, 2005.

[30] K. Kummamuru, R. Krishnapuram, and R. Agrawal. On learning assymetric dissimilarity measures. In *ICDM*, 2005.

[31] D. Papadias, Y. Tao, G. Fu, and B. Seeger. An optimal and progressive algorithm for skyline queries. In *SIGMOD*, pages 467–478, 2003.

[32] M. Richardson and P. Domingos. The intelligent surfer: Probabilistic combination of link and content information in pagerank. In *NIPS*, pages 1441–1448, 2001.

[33] M. Schultz and T. Joachims. Learning a distance metric from relative comparisons. In *NIPS*, 2003.

[34] M. Yannakakis. Edge-deletion problems. *SIAM J. Comput.*, 10(2):297–309, 1981.