PerK: Personalized Keyword Search in Relational Databases through Preferences

Kostas Stefanidis
Dept. of Computer Science
University of Ioannina, Greece
kstef@cs.uoi.gr

Marina Drosou
Dept. of Computer Science
University of Ioannina, Greece
mdrosou@cs.uoi.gr

Evaggelia Pitoura
Dept. of Computer Science
University of Ioannina, Greece
pitoura@cs.uoi.gr

ABSTRACT

Keyword-based search in relational databases allows users to discover relevant information without knowing the database schema or using complicated queries. However, such searches may return an overwhelming number of results, often loosely related to the user intent. In this paper, we propose personalizing keyword database search by utilizing user preferences. Query results are ranked based on both their relevance to the query and their preference degree for the user. To further increase the quality of results, we consider two new metrics that evaluate the goodness of the result as a set, namely coverage of many user interests and content diversity. We present an algorithm for processing preference queries that uses the preferential order between keywords to direct the joining of relevant tuples from multiple relations. We then show how to reduce the complexity of this algorithm by sharing computational steps. Finally, we report evaluation results of the efficiency and effectiveness of our approach.

1. INTRODUCTION

Keyword-based search is very popular because it allows users to express their information needs without either being aware of the underlying structure of the data or using a query language. In relational databases, existing keyword search approaches exploit the database schema or the given database instance to retrieve tuples relevant to the keywords of the query. For example, consider the movie database instance shown in Figure 1. Then, the results of the keyword query $Q = \{thriller, B.\ Pitt\}$ are the *thriller* movies *Twelve Monkeys* and *Seven* both with *B. Pitt*.

Keyword search is intrinsically ambiguous. Given the abundance of available information, exploring the contents of a database is a complex procedure that may return a huge volume of data. Still, users would like to retrieve only a small piece of it, namely the most relevant to their interests. Previous approaches for ranking the results of keyword search include, among others, adapting IR-style document relevance ranking strategies (e.g. [18]) and exploiting the link structure of the database (e.g. [19, 6, 9]). In this paper, we propose personalizing database keyword search, so that different users receive different results based on their personal interests. To this end, the proposed model exploits user preferences for ranking keyword results.

In our model, preferences express a user *choice* that holds under a specific *context*, where both context and choice are specified through keywords. For example, consider the following two preferences: ($\{thriller\}$, *G. Oldman* $\succ$ *W. Allen*) and ($\{comedy\}$, *W. Allen* $\succ$ *G. Oldman*). The first preference denotes that in the context of *thriller* movies, the user prefers *G. Oldman* over *W. Allen*, whereas the latter, that in the context of *comedies*, the user prefers *W. Allen* over *G. Oldman*. Such preferences may be specified in an ad-hoc manner when the user submits a query or they may be stored in a general user profile. Preferences may also be created automatically based on explicit or implicit user feedback (e.g. [12, 20]) or on the popularity of specific keyword combinations (e.g. [17, 4]). For example, the first preference may be induced by the fact that the keywords *thriller* and *G. Oldman* co-occur in the query log more often than the keywords *thriller* and *W. Allen*.

Given a set of preferences, we would like to personalize a keyword query Q by ranking its results in an order compatible with the order expressed in the user choices for context Q. For example, in the results of the query $Q = \{thriller\}$, movies related to *G. Oldman* should precede those related to *W. Allen*. To formalize this requirement, we consider expansions of query Q with the set of keywords appearing in the user choices for context Q. For instance, for the query $Q = \{thriller\}$, we use the queries $Q_1 = \{thriller, G.\ Oldman\}$ and $Q_2 = \{thriller, W.\ Allen\}$. We project the order induced by the user choices among the results of these queries to produce an order among the results of the original query Q.

Since keyword search is often best-effort, given a constraint k on the number of results, we would like to combine the order of results as indicated by the user preferences with their relevance to the query. Besides preferences and relevance, we also consider the set of the k results as a whole and seek to increase the overall value of this set to the users. Specifically, we aim at selecting the k most representative among the relevant and preferred results, i.e. these results that both cover different preferences and have different content. In general, such result diversification, i.e. selecting items that differ from each other, has been shown to increase user satisfaction [36, 34].

We propose a number of algorithms for computing the top-k results. For generating results that follow the preference order, we rely on applying the *winnow operator* [11, 32] on various levels to retrieve the most preferable choices at each level. Then, we introduce a *sharing-results* keyword query processing algorithm, that exploits the fact that the results of a keyword query are related with the results of its superset queries, to avoid redundant computations. Finally, we propose an algorithm that works in conjunction with the multi-level winnow and the sharing-results algorithm to compute the top-k representative results.

EDBT 2010, March 22–26, 2010, Lausanne, Switzerland.

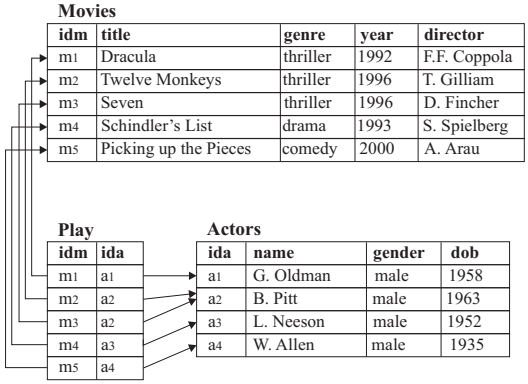


Figure 1: Database instance.

In summary, this paper makes the following contributions:

- it proposes personalizing keyword search through user preferences and provides a formal model for integrating preferential ranking with database keyword search,
- it combines multiple criteria for the quality of the results that include the relevance and the degree of preference of each individual result as well as the coverage and diversity of the set of results as a whole,
- it presents efficient algorithms for the computation of the top-k representative results.

We have evaluated both the efficiency and effectiveness of our approach. Our performance results show that the sharing-results algorithm improves the execution time over the baseline one by 90%. Furthermore, the overall overhead for preference expansion and diversification is reasonable (around 30% in most cases). Our usability results indicate that users receive results more interesting to them when preferences are used.

The rest of this paper is organized as follows. In Section 2, we introduce our contextual keyword preference model. In Section 3, we explore the desired properties of search results and define the top-k representative ones, while in Section 4, we propose algorithms for preferential keyword query processing within relational databases. In Section 5, we discuss a number of extensions and in Section 6, we present our evaluation results. Section 7 describes related work and finally, Section 8 concludes the paper.

2. MODEL

We start this section with a short introduction to keyword search in databases. Then, we present our model of preferences and personalized keyword search.

2.1 Preliminaries

Most approaches to keyword search (e.g. [19, 6]) exploit the dependencies in the database schema for answering keyword queries. Consider a database $\mathcal{R}$ with n relations $R_1, R_2, \ldots, R_n$. The *schema graph* $\mathcal{G}_D$ is a directed graph capturing the foreign key relationships in the schema. $\mathcal{G}_D$ has one node for each relation R_i and an edge $R_i \rightarrow R_j$, if and only if, R_i has a set of foreign key attributes referring to the primary key attributes of R_j. We refer to the undirected version of the schema graph as $\mathcal{G}_U$.

Let W be the potentially infinite set of all keywords. A keyword query Q consists of a set of keywords, i.e. $Q \subseteq W$. Typically, the result of a keyword query is defined with regards to *joining trees of tuples* (JTTs), which are trees of tuples connected through primary to foreign key dependencies [19, 6, 9].

DEFINITION 1 (JOINING TREE OF TUPLES (JTT)). *Given an undirected schema graph $\mathcal{G}_U$, a joining tree of tuples (JTT) is a tree of tuples T, such that, for each pair of adjacent tuples t_i, t_j in T, $t_i \in R_i$, $t_j \in R_j$, there is an edge $(R_i, R_j) \in \mathcal{G}_U$ and it holds that $(t_i \bowtie t_j) \in (R_i \bowtie R_j)$.*

Total JTT: A JTT T is *total* for a keyword query Q, if and only if, every keyword of Q is contained in at least one tuple of T.

Minimal JTT: A JTT T that is total for a keyword query Q is also *minimal* for Q, if and only if, we cannot remove a tuple from T and get a total JTT for Q.

We can now define the result of a keyword query as follows:

DEFINITION 2 (QUERY RESULT). *Given a keyword query Q, the result $Res(Q)$ of Q is the set of all JTTs that are both total and minimal for Q.*

The size of a JTT is equal to the number of its tuples, i.e. the number of nodes in the tree, which is one more than the number of joins. For example, for the database of Figure 1, the result of the keyword query Q = {*thriller*, *B. Pitt*} consists of the JTTs: (i) (m_2, *Twelve Monkeys, thriller, 1996, T. Gilliam*) $-$ (m_2, a_2) $-$ (a_2, *B. Pitt, male, 1963*) and (ii) (m_3, *Seven, thriller, 1996, D. Fincher*) $-$ (m_3, a_2) $-$ (a_2, *B. Pitt, male, 1963*), both of size equal to 3.

2.2 Keyword Preference Model

Keyword queries are very general and their result may include a large number of JTTs. We propose personalizing such results by incorporating preferences.

DEFINITION 3 (CONTEXTUAL KEYWORD PREFERENCE). *A contextual keyword preference cp is a pair cp = (C, $w_i \succ w_j$), where $C \subseteq W$ and $w_i, w_j \in W$. We also write $w_i \succ_C w_j$.*

The intuitive meaning of a contextual keyword preference, or simply preference, is that, when all keywords in context C are present, results involving keyword w_i are preferred over those involving keyword w_j. We refer to $w_i \succ_C w_j$ as the choice part of the preference. For example, consider the preference *cp* = ({*thriller, B. Pitt*}, *T. Gilliam* $\succ$ *D. Fincher*). Preference *cp* indicates that, in the case of *thrillers* and *B. Pitt*, movies related to *T. Gilliam* are preferred over those related to *D. Fincher*.

Note that we interpret context using AND semantics. This means that a choice holds only if all the keywords of the context part are present (both *thriller* and *B. Pitt* in our example). OR semantics can be achieved by having two or more preferences with the same choice part (for instance, in our example, one for *thriller* and one for *B. Pitt*).

We call the preferences for which the context part is empty, i.e. $C = \{\}$, *context-free* keyword preferences. Context-free keyword preferences may be seen as preferences that hold independently of context. For example, the preference ({}, *thriller* $\succ$ *drama*) indicates that *thrillers* are preferred over *dramas* unconditionally.

We call the set of all preferences defined by a user, user profile, or simply *profile*. Let P be a profile, we use P_C to denote the set of preferences with context C and W_C to denote the set of keywords that appear in the choices of P_C. We call the keywords in W_C *choice keywords* for C.

We provide next the formal definition of dominance.

DEFINITION 4 (DIRECT PREFERENTIAL DOMINATION). *Given a keyword query Q and a profile P, let T_i, T_j be two JTTs total for Q. We say that T_i directly dominates T_j under P_Q, $T_i \succ_{P_Q} T_j$, if and only if, $\exists w_i$ in T_i, such that, $\nexists w_j$ in T_j with $w_j \succ_Q w_i$ and $w_i, w_j \in W_Q$.*

The motivation for this specific formulation of the definition of dominance is twofold. First, we want to favor JTTs that include at least one choice keyword over those that do not include any such keyword. Second, in the case of two JTTs that contain many choice keywords, we want to favor the JTT that contains the most preferred

one among them. To clarify this, consider the following example. Assume the query $Q = \{w_q\}$, the choice keywords w_1, w_2, w_3, w_4 and the preferences $(\{w_q\}, w_1 \succ w_2)$, $(\{w_q\}, w_2 \succ w_3)$, $(\{w_q\}, w_4 \succ w_2)$. Let T_1, T_2 be two JTTs in the result set of Q, where T_1 contains, among others, the keywords w_q, w_1, w_3 and T_2 the keywords w_q and w_2. Then, based on Definition 4, although T_1 contains the keyword w_3 that is less preferable than w_2 contained in T_2, T_1 directly dominates T_2, because T_1 contains w_1 which is the most preferred keyword among them.

In general, direct dominance $\succ_{P_Q}$ defines an order among the JTTs that contain all keywords in Q. Note that it is possible that, for two JTTs T_1, T_2, both $T_1 \succ_{P_Q} T_2$ and $T_2 \succ_{P_Q} T_1$ hold. For instance, in the above example, assume T_1 with w_q and w_1 and T_2 with w_q and w_4. We consider such JTTs to be equally preferred. It is also possible that neither $T_1 \succ_{P_Q} T_2$ nor $T_2 \succ_{P_Q} T_1$ holds. This is the case when none of the JTTs contain any choice keywords. Such JTTs are incomparable; we discuss next how we can order them.

2.3 Extending Dominance

Definition 4 can be used to order by dominance those JTTs in the query result that contain choice keywords. For example, given the preference ({*thriller*}, *F. F. Coppola* $\succ$ *T. Gilliam*), for the query Q = {*thriller*}, the JTT $T_1 = (m_1$, *Dracula, thriller, 1992, F. F. Coppola*) directly dominates the JTT $T_2 = (m_2$, *Twelve Monkeys, thriller, 1996, T. Gilliam*). However, we cannot order results that may contain choice keywords indirectly through joins. For example, given the preference ({*thriller*}, *G. Oldman* $\succ$ *B. Pitt*) and the same query Q = {*thriller*}, now T_1 and T_2 do not contain any choice keywords and thus are incomparable, whereas again T_1 should be preferred over T_2 since it is a thriller movie related to *G. Oldman*, while T_2 is related to *B. Pitt*.

We capture such indirect dominance through the notion of a JTT projection. Intuitively, a JTT T_i indirectly dominates a JTT T_j, if T_i is the projection of some JTT that directly dominates the JTTs whose projection is T_j.

Projected JTT: Assume a keyword query Q and let T_i, T_j be two JTTs. T_j is a projected JTT of T_i for Q, if and only if, T_j is a subtree of T_i that is total and minimal for Q, that is, $T_j \in Res(Q)$. The set of the projected JTTs of T_i for Q is denoted by $project_Q(T_i)$.

For example, assume the query Q = {*thriller*}. The JTT (m_1, *Dracula, thriller, 1992, F. F. Coppola*) is a projected JTT of (m_1, *Dracula, thriller, 1992, F. F. Coppola*) $-$ (m_1, a_1) $-$ (a_1, *G. Oldman, male, 1958*) for Q.

We can construct the projected JTTs of a JTT T by appropriately removing nodes from T as follows. A leaf node of T is called *secondary* with respect to Q, if it contains a keyword in Q that is also contained in some other node of T. All projected JTTs for T can be produced from T by removing secondary nodes one by one till none remains.

The following set is useful. It contains *exactly* the minimal JTTs that include all keywords in Q and at least one keyword in W_Q.

DEFINITION 5 (PREFERENTIAL QUERY RESULT). *Given a keyword query Q and a profile P, the preferential query result $PRes(Q, P)$ is the set of all JTTs that are both total and minimal for at least one of the queries $Q \bigcup \{w_i\}$, $w_i \in W_Q$.*

Now, we can define indirect dominance as follows:

DEFINITION 6 (INDIRECT PREFERENTIAL DOMINATION). *Given a keyword query Q and a profile P, let T_i, T_j be two JTTs total for Q. We say that T_i indirectly dominates T_j under P_Q, $T_i \succ\!\succ_{P_Q} T_j$, if there is a JTT $T_i' \in PRes(Q, P)$, such that, $T_i \in project_Q(T_i')$ and there is no joining tree of tuples $T_j' \in PRes(Q, P)$, such that, $T_j \in project_Q(T_j')$ and $T_j' \succ_{P_Q} T_i'$.*

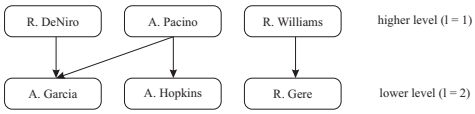


Figure 2: The graph of choices $G_{P_{\{thriller, F.F.Coppola\}}}$.

Note that the indirect dominance relation is not a superset of direct dominance, that is, $T_i \succ_{P_Q} T_j \not\Rightarrow T_i \succ\!\succ_{P_Q} T_j$. To see this, consider the case where T_i contains a choice keyword that precedes those in T_j but T_j belongs to the project of a JTT that contains an even more preferred keyword.

Our goal in defining indirect preferential dominance is to impose an ordering over the results that will follow the preferences given by the users exactly. Thus, a result that is even only "distantly" related to a choice keyword (i.e. through many joins) is preferred over a result that is more closely related to a less preferred choice keyword. We shall introduce issues of relevance and multi-criteria ranking later in the paper.

2.4 Processing Dominance

Given a query Q, we would like to generate its results in order of indirect dominance. To achieve this, we use the fact that, in general, the trees in the result of $Q \bigcup \{w_i\}$ directly dominate the trees in the result of $Q \bigcup \{w_j\}$, for $w_i \succ_Q w_j$. This suggests that the order for generating the results for a query Q should follow the order $\succ_Q$ among the choice keywords in W_Q. We describe next, how to organize the choice keywords to achieve this.

Let P be a profile, C a context and P_C the related contextual preferences in P. We organize the choice keywords in W_C using a directed graph G_{P_C} for P_C, referred to as *graph of choices* for P_C. G_{P_C} has one node for each keyword $w_i \in W_C$ and an edge from the node representing w_i to the node representing w_j, if and only if, it holds that $w_i \succ_C w_j$ and $\nexists w_r$, such that, $w_i \succ_C w_r$ and $w_r \succ_C w_j$. For example, consider the preferences for C = {*thriller, F. F. Coppola*}: cp_1 = (C, *R. DeNiro* $\succ$ *A. Garcia*), cp_2 = (C, *A. Pacino* $\succ$ *A. Garcia*), cp_3 = (C, *A. Pacino* $\succ$ *A. Hopkins*) and cp_4 = (C, *R. Williams* $\succ$ *R. Gere*). The graph of choices for this set of preferences is depicted in Figure 2.

To extract from G_{P_C} the set of the most preferred keywords, we apply the *multiple level winnow operator* [11, 32]. This operator retrieves the keywords appearing in G_{P_C} in order of preference. Specifically, at level 1, $win_{P_C}(1) = \{w_i \in W_C \mid \nexists w_j \in W_C, w_j \succ_C w_i\}$. For subsequent applications at level l, $l > 1$, it holds, $win_{P_C}(l) = \{w_i \in W_C \mid \nexists w_j \in (W_C - \bigcup_{r=1}^{l-1} win_{P_C}(r))$ with $w_j \succ_C w_i\}$.

In the following, we assume that the preference relation $\succ_C$ defined over the keywords in W_C is a strict partial order. This means that it is irreflexive, asymmetric and transitive. Irreflexivity and asymmetry are intuitive, while transitivity allows users to define priorities among keywords without the need of specifying relationships between all possible pairs. Strict partial order ensures that there are no cycles in preferences, since that would violate irreflexivity.

Since the relation $\succ_C$ is acyclic, this ordering of keywords corresponds to a topological sort of G_{P_C}. Therefore, we traverse the graph of choices G_{P_C} in levels (Algorithm 1) and at each level, we return the keywords of the nodes with no incoming edges. For example, consider the graph of choices of Figure 2 for C = {*thriller, F. F. Coppola*}. Then, $win_{P_C}(1)$ = {*R. DeNiro, A. Pacino, R. Williams*}, while $win_{P_C}(2)$ = {*A. Garcia, A. Hopkins, R. Gere*}.

Let T be a JTT that belongs to $PRes(Q, P)$. To encapsulate the

Algorithm 1 Multiple Level Winnow Algorithm

Input: A graph of choices $G_{P_C} = (V_G, E_G)$.
Output: The sets $win_{P_C}(l)$ for the levels l.

```
1: begin
2: winnow_result: empty list;
3: l = 1;
4: while V_G not empty do
5:    for all w_i ∈ V_G with no incoming edges in E_G do
6:       win_{P_C}(l) = win_{P_C}(l) ⋃ {w_i};
7:    end for
8:    Add win_{P_C}(l) to winnow_result;
9:    V_G = V_G − win_{P_C}(l);
10:   for all edges e = (w_i, w_j) with w_i in win_{P_C}(l) do
11:      E_G = E_G − e;
12:   end for
13:   l++;
14: end while
15: return winnow_result;
16: end
```

preference order of T with regards to Q and P, we associate with T a value, called $dorder(T, Q, P)$, equal to the minimum winnow level l over all choice keywords $w_i \in W_Q$ that appear in T. Then:

PROPOSITION 1. *Let T_i, T_j be two JTTs, T_i, $T_j \in PRes(Q, P)$, such that, $dorder(T_i, Q, P) < dorder(T_j, Q, P)$. Then, T_j does not directly dominate T_i under P_Q.*

PROOF. For the purpose of contradiction, assume that $T_j \succ_{P_Q} T_i$. Then, $\exists w_j$ in T_j, such that, $\nexists w_i$ in T_i with $w_i \succ_Q w_j$, which means that $dorder(T_i, Q, P) \geq dorder(T_j, Q, P)$, which is a contradiction. □

Thus, by executing the queries $Q \bigcup \{w_1\}, \ldots, Q \bigcup \{w_m\}$, where $\{w_1, \ldots, w_m\}$ are the keywords retrieved by the multiple level winnow operator, in that order, we retrieve the JTTs of $PRes(Q, P)$ in an order compatible with the direct dominance relation among them. Given, for example, the query Q = {*thriller, F. F. Coppola*} and the preferences cp_1, cp_2, cp_3 and cp_4, we report first the JTTs in the results of $Q \cup$ {*R. DeNiro*}, $Q \cup$ {*A. Pacino*}, $Q \cup$ {*R. Williams*} and then, those for $Q \cup$ {*A. Garcia*}, $Q \cup$ {*A. Hopkins*}, $Q \cup$ {*R. Gere*}.

By taking the projection of these JTTs in that order, and removing duplicate appearances of the same trees, we take results in $Res(Q)$ in the correct indirect dominance order. Note that a projected result may appear twice as output since it may be related indirectly, i.e. through joins, with more than one choice keyword.

To see that by projecting the JTTs we get the results in $Res(Q)$ ordered by indirect dominance, let T be a JTT that belongs to $Res(Q)$. We define the indirect order of T, $iorder(T, Q, P)$, to capture its indirect dominance with respect to Q as follows: $iorder(T, Q, P)$ is the minimum $dorder(T', Q, P)$ among all T', such that, $T \in project_Q(T')$ and ∞ if there is no such T'. It holds:

THEOREM 1. *Let T_i, T_j be two JTTs, T_i, $T_j \in Res(Q)$, such that, $iorder(T_i, Q, P) < iorder(T_j, Q, P)$. Then, T_j does not indirectly dominate T_i under Q.*

PROOF. Assume that $T_j \succ\succ_{P_Q} T_i$. Then $\exists T'_j \in PRes(Q, P)$, such that, $T_j \in project_Q(T'_j)$ and $\nexists T'_i \in PRes(Q, P)$, such that, $T_i \in project_Q(T'_i)$ with $T'_i \succ_{P_Q} T'_j$. Since T_i is a subtree of T'_i, $\neg(T_i \succ_{P_Q} T'_j)$ (1). Also, since $iorder(T_i, Q, P) < iorder(T_j, Q, P)$ and $T_j \in project_Q(T'_j)$, T'_j cannot contain any keyword that is preferred over the keywords of T_i. Therefore, $\neg(T'_j \succ_{P_Q} T_i)$ (2). Since T'_j contains at least one choice keyword, (1) and (2) cannot hold simultaneously, which is a contradiction. □

Note here that there may be results in $Res(Q)$ that we do not get by projection. Those do not indirectly dominate any result but are indirectly dominated by those that we have gotten by projection.

THEOREM 2. *Let $S = \bigcup_r project_Q(T_r)$, $\forall T_r \in PRes(Q, P)$, and T_i be a JTT, such that, $T_i \in Res(Q) \backslash S$. Then, $\forall T_j \in S$, it holds that (i) $T_j \succ\succ_{P_Q} T_i$ and (ii) $\neg(T_i \succ\succ_{P_Q} T_j)$.*

PROOF. Since $T_i \notin S$, there is no T'_i, $T_i \in project_Q(T'_i)$, such that, T'_i contains a choice keyword of W_Q. However, for every $T_j \in S$ there is at least one T'_j, $T_j \in project_Q(T'_j)$, such that, T'_j contains at least a choice keyword of W_Q. Therefore, according to Definition 6, both (i) and (ii) hold. □

We can present to the user the projected result or the original JTT in $PRes(Q, P)$, which is not minimal but provides an explanation of why its projected tree in $Res(Q)$ was ordered this way. For instance, for the query Q = {*thriller*}, the preference ({*thriller*}, *G. Oldman* $\succ$ *B. Pitt*) and the database instance in Figure 1, we could either present to the user as top result the JTT (m_1, *Dracula, thriller, 1992, F. F. Coppola*) $-$ (m_1, a_1) $-$ (a_1, *G. Oldman, male, 1958*) that belongs to $PRes(Q, P)$ or its projected JTT (m_1, *Dracula, thriller, 1992, F. F. Coppola*) that belongs to $Res(Q)$.

3. TOP-K PERSONALIZED RESULTS

In general, keyword search is best effort. For achieving useful results, dominance needs to be combined with other criteria. We distinguish between two types of properties that affect the goodness of the result: (i) properties that refer to each individual JTT in the result and (ii) properties that refer to the result as a whole. The first type includes preferential dominance and relevance, while the latter includes coverage of user interests and diversity.

3.1 Result Goodness

Each individual JTT T total for a query Q is characterized by its dominance with regards to a profile, denoted $iorder(T, Q, P)$. In addition, there has been a lot of work on ranking JTTs based on their relevance to the query. A natural characterization of the relevance of a JTT (e.g. [19, 6]) is its size: the smaller the size of the tree, the smaller the number of the corresponding joins, thus the larger its relevance. The relevance of a JTT can also be computed based on the importance of its tuples. For example, [9] assigns scores to JTTs based on the prestige of their tuples, i.e. the number of their neighbors or the strength of their relationships with other tuples, while [18] adapts IR-style document relevance ranking. In the following, we do not restrict to a specific definition of relevance, but instead just assume that each individual JTT T is also characterized by a degree of relevance, denoted $relevance(T, Q)$.

Apart from properties of each individual JTT, to ensure user satisfaction by personalized search, it is also important for the whole set of results to exhibit some desired properties. In this paper, we consider covering many user interests and avoiding redundant information.

To understand coverage, consider the graph of choices in Figure 2. JTTs for the query Q ={*thriller, F. F. Copolla*} that include *R. DeNiro* and *R. Williams* have the same degree of dominance and assume, for the purposes of this example, that they also have the same relevance. Still, we would expect that a good result does not only include JTTs (i.e. movies) that cover the preference on *R. DeNiro* but also JTTs that cover the preference on *R. Williams* and perhaps other choices as well. To capture this requirement, we define the *coverage* of a set S of JTTs with regards to a query Q as the percentage of choice keywords in W_Q that appear in S. Formally:

DEFINITION 7 (COVERAGE). *Given a query Q, a profile P and a set $S = \{T_1, \ldots, T_z\}$ of JTTs that are total for Q, the coverage of S for Q and P is defined as:*

$$coverage(S, Q, P) = \frac{\left|\bigcup_{i=1}^{z}(W_Q \cap keywords(T_i))\right|}{|W_Q|},$$

where $keywords(T_i)$ is the set of keywords in T_i.

High coverage ensures that the user will find many interesting results among the retrieved ones. However, many times, two JTTs may contain the same or very similar information, even if they are computed for different choice keywords. To avoid such redundant information, we opt to provide users with results that exhibit some diversity, i.e. they do not contain overlapping information. For quantifying the overlap between two JTTs, we use a Jaccard-based definition of distance, which measures dissimilarity between the tuples that form these trees. Given two JTTs T_i, T_j consisting of the sets of tuples A, B respectively, the distance between T_i and T_j is: $d(T_i, T_j) = 1 - \frac{|A \cap B|}{|A \cup B|}$. We have considered other types of distances as well, but this is simple, relatively fast to compute and provides a good indication of the overlapping content of the two trees.

To measure the overall diversity of a set of JTTs, we next define their set diversity based on their distances from each other. A number of different definitions for set diversity have been proposed in the context of recommender systems; here we model diversity as the average distance of all pairs of elements in the set [35].

DEFINITION 8 (SET DIVERSITY). *Given a set S of z JTTs, $S = \{T_1, \ldots, T_z\}$, the set diversity of S is:*

$$diversity(S) = \frac{\sum_{i=1}^{z} \sum_{j>i}^{z} d(T_i, T_j)}{(z-1)z/2}.$$

To summarize, a "good" result S for a query Q includes JTTs that are preferred and relevant, covers many choices and is diverse.

3.2 Top-k Result Selection

Given a restriction k on the size of the result, we would like to provide users with k highly preferable and relevant results that also as a whole cover many of their choices and exhibit low redundancy. To achieve this, we resort to the following algorithm that offers us the flexibility of fine-tuning the importance of each of the criteria in selecting the top-k results.

For a query Q, we use $Res_s(Q)$ to denote the set of JTTs with relevance greater than a threshold s. Given a query Q and a profile P, let l be the maximum winnow level. For $1 \leq r \leq l$, let $Z^r = \bigcup_{w_j \in win_{P_Q}(r)} Res_s(Q \cup \{w_j\})$. Also, let $Z^{l+1} = Res_s(Q) \setminus \bigcup_{T_e \in PRes(Q,P)} project_Q(T_e)$. We want more preferred keywords, that is, the ones corresponding to small winnow values, to contribute more trees to the top-k results than less preferred ones. The number of trees offered by each level i is captured by $\mathcal{F}(i)$, where $\mathcal{F}$ is a monotonically decreasing function with $\sum_{i=1}^{l+1} \mathcal{F}(i) = k$. Each Z^i contributes $\mathcal{F}(i)$ JTTs. For $1 \leq i \leq l$, the contributed JTTs are uniformly distributed among the keywords of level i to increase coverage.

Among the many possible combinations of k trees that satisfy the constraints imposed by $\mathcal{F}$, we choose the one with the most diverse results. Next, we define the top-k JTTs.

DEFINITION 9 (TOP-k JTTS). *Given a keyword query Q, a profile P, a relevance threshold s and the sets of results $\{Z^1, \ldots, Z^l, Z^{l+1}\}$ with $|Z^1|+\ldots+|Z^l|+|Z^{l+1}| = m$, the top-$k$ JTTs, $k < m$, is the set S^* for which:*

$$S^* = \underset{\substack{S \subseteq \bigcup_{i=1}^{l+1} Z^i \\ |S|=k}}{\operatorname{argmax}} \; diversity(S),$$

such that, Z^i contributes $\mathcal{F}(i)$ JTTs to S^*, which, for $1 \leq i \leq l$, are uniformly distributed among the keywords of winnow level i and $\mathcal{F}$ is a monotonically decreasing function with $\sum_{i=1}^{l+1} \mathcal{F}(i) = k$.

There are two basic tuning parameters: the function $\mathcal{F}$ and the threshold s. Dominance, coverage and relevance depend on how quickly $\mathcal{F}$ decreases. A high decrease rate leads to keywords from fewer winnow levels contributing to the final result. This means that coverage will generally decrease. However, at the same time, the average dominance will increase, since the returned results correspond to high winnow levels only. For example, if a user is primarily interested in dominant results, we retrieve k JTTs corresponding to keywords retrieved by $win_{P_Q}(1)$ by setting, for example, $\mathcal{F}(1) = k$, and $\mathcal{F}(i) = 0$, for $i > 1$. A low decrease rate of $\mathcal{F}$ means that less trees will be retrieved from each winnow level, so we can retrieve the most relevant ones. Relevance is also calibrated through the selection of the relevance threshold, s. If relevance is more important than dominance, a large value for the relevance threshold in conjunction with an appropriate $\mathcal{F}$ will result in retrieving the k JTTs that have the largest degrees of relevance, including those in Z^{l+1} that do no have any relation with any choice keyword. Diversity is calibrated through s that determines the number m of candidate trees out of which to select the k most diverse ones.

4. QUERY PROCESSING

In this section, we present our algorithms for processing personalized keyword queries. Section 4.1 presents some background, while in Section 4.2, we first present a baseline algorithm for processing keyword queries and then introduce an enhancement that reuses computational steps to improve performance. In Section 4.3, we propose an algorithm for computing top-k results.

4.1 Background

We use our movies example (Figure 1) to briefly describe basic ideas of existing keyword query processing. For instance, consider the query Q = {*thriller*, *B. Pitt*}. The corresponding result consists of the JTTs: (i) (m_2, *Twelve Monkeys, thriller, 1996, T. Gilliam*) $-$ (m_2, a_2) $-$ (a_2, *B. Pitt, male, 1963*) and (ii) (m_3, *Seven, thriller, 1996, D. Fincher*) $-$ (m_3, a_2) $-$ (a_2, *B. Pitt, male, 1963*). Each JTT corresponds to a tree at schema level. For example, both of the above trees correspond to the schema level tree $Movies^{\{thriller\}} - Play^{\{\}} - Actors^{\{B.Pitt\}}$, where each R_i^X consists of the tuples of R_i that contain all keywords of X and no other keyword of Q. Such sets are called *tuple sets* and the schema level trees are called *joining trees of tuple sets* (JTSs).

Several algorithms in the research literature aim at constructing such trees of tuple sets for a query Q as an intermediate step of the computation of the final results (e.g. [19, 6]). In the following, we adopt the approach of [19], in which all JTSs with size up to s are constructed (in this case, a JTT's size determines its relevance). In particular, given a query Q, all possible tuple sets R_i^X are computed, where $R_i^X = \{t \mid t \in R_i \wedge \forall w_x \in X, t \text{ contains } w_x \wedge \forall w_y \in Q \backslash X, t \text{ does not contain } w_y\}$. After selecting a random query keyword w_z, all tuple sets R_i^X for which $w_z \in X$ are located. These are the initial JTSs with only one node. Then, these trees are expanded either by adding a tuple set that contains at least another query keyword or a tuple set for which $X = \{\}$ (free tuple set). These trees can be further expanded. JTSs that contain all query keywords are returned, while JTSs of the form $R_i^X - R_j^{\{\}} - R_i^Y$, where an edge $R_j \rightarrow R_i$ exists in the schema graph, are pruned, since JTTs produced by them have more than one occurrence of the same tuple for every instance of the database.

4.2 Processing Preferential Queries

In this section, we present algorithms for computing the preferential results of a query, ranked in an order compatible with preferential dominance.

4.2.1 Baseline Approach

The *Baseline JTS Algorithm* (Algorithm 2) constructs in levels the sets of JTSs for the queries $Q \cup \{w_i\}, \forall w_i \in win_{P_Q}(l)$, start-

Algorithm 2 Baseline JTS Algorithm

Input: A query Q, a profile P, a schema graph $\mathcal{G}_U$ and a size s.
Output: A list JTList of JTSs with size up to s for the queries $Q \bigcup \{w_i\}$, $\forall w_i \in W_{P_Q}$.

1: **begin**
2: $Queue$: queue of JTSs;
3: $JTList$: empty list;
4: l = 1;
5: **while** unmarked keywords exist in W_{P_Q} **do**
6: Compute the set of keywords $win_{P_Q}(l)$;
7: **for** each $w_z \in win_{P_Q}(l)$ **do**
8: Mark w_z;
9: Compute the tuple sets R_i^X for $Q \bigcup \{w_z\}$;
10: Select a keyword $w_t \in Q \bigcup \{w_z\}$;
11: **for** each R_i^X, $1 \leq i \leq n$, such that, $w_t \in X$ **do**
12: Insert R_i^X into $Queue$;
13: **end for**
14: **while** $Queue \neq \emptyset$ **do**
15: Remove the head B from $Queue$;
16: **if** B satisfies the pruning rule **then**
17: Ignore B;
18: **else if** $keys(B) = Q \bigcup \{w_z\}$ **then**
19: Insert B into $JTList$;
20: **else**
21: **for** each R_i^X, such that, there is an R_j^Y in B and R_i is adjacent to R_j in $\mathcal{G}_U$ **do**
22: **if** ($X = \{\}$ OR $X - keys(B) \neq \emptyset$) AND (size of $B < s$) **then**
23: Expand B to include R_i^X;
24: Insert the updated B into $Queue$;
25: **end if**
26: **end for**
27: **end if**
28: **end while**
29: **end for**
30: l++;
31: **end while**
32: **return** $JTList$;
33: **end**

ing with $l = 1$, i.e. the level with the most preferred keywords. This way, all JTTs constructed for JTSs produced at level l are retrieved before the JTTs of the trees of tuple sets produced at level l+1. Algorithm 2 terminates when all the JTSs for queries $Q \cup \{w_i\}, \forall w_i \in W_{P_Q}$, have been computed. (In Algorithm 2, we use the notation $keys(B)$ to refer to the query keywords contained in a JTS B.)

Based on the completeness theorem of the algorithm introduced in [19] for computing the JTSs, Theorem 3 proves the completeness of Algorithm 2.

THEOREM 3 (COMPLETENESS). *Every JTT of size s_i that belongs to the preferential query result of a keyword query Q is produced by a JTS of size s_i that is constructed by the Baseline JTS Algorithm.*

PROOF. Given a query Q and a profile P, the *Baseline JTS Algorithm* constructs independently the JTSs for each query $Q \cup \{w_t\}, \forall w_t \in W_{P_C}$ (lines 8-27). Since for each query the algorithm returns the trees of tuple sets that construct every JTT that belongs to the corresponding result, every JTT that belongs to $PRes(Q, P)$ is produced by the JTSs constructed by Algorithm 2 as well. □

4.2.2 Result Sharing

Based on the observation that the JTSs for Q may already contain in their tuple sets the additional keyword w_t of a query $Q_t \in KQ$, where KQ contains the queries $Q_t = Q \cup \{w_t\}, \forall w_t \in W_{P_Q}$, we employ such trees to construct those for Q_t. To do this, the *Sharing JTS Algorithm* (Algorithm 3) constructs first the JTSs for Q using a selected keyword $w_r \in Q$ based on the tuple sets R_i^X for Q (lines 3-5). Then, for each Q_t, we recompute its tuple sets by partitioning each R_i^X for Q into two tuple sets for Q_t: R_i^X that contains the tuples with only the keywords X and $R_i^{X \cup \{w_t\}}$ that contains the tuples with only the keywords $X \cup \{w_t\}$ (lines 11-13). Using the JTSs for Q and the tuple sets for Q_t, we produce all combinations of trees of tuple sets (lines 14-17) that will be used next to construct the final JTSs for Q_t. For example, given the JTS for Q R_i^X - R_j^Y, we produce the following JTSs for Q_t: R_i^X - R_j^Y, $R_i^{X \cup \{w_t\}}$ - R_j^Y, R_i^X - $R_j^{Y \cup \{w_t\}}$ and $R_i^{X \cup \{w_t\}}$ - $R_j^{X \cup \{w_t\}}$. Note that, such a JTS is constructed only if all of its tuples sets are non-empty. The JTSs that contain all keywords of Q_t are returned. The rest of them are expanded as in Algorithm 2 (lines 33-42).

Since for a query Q Algorithm 2 does not construct JTSs of the form $R_i^{\{w_k\}}$ - $R_j^{\{w_k\}}$, the procedure described above does not construct for Q_t JTSs of the form $R_i^{\{w_k\}}$ - $R_j^{\{w_k, w_t\}}$. The same also holds for the JTSs that connect $R_i^{\{w_k\}}$, $R_j^{\{w_k, w_t\}}$ via free tuple sets. To overcome this, we construct all such trees from scratch (lines 18-32) and then expand them as before (lines 33-42). Theorem 4 proves the completeness of Algorithm 3.

THEOREM 4 (COMPLETENESS). *Every JTT of size s_i that belongs to the preferential query result of a keyword query Q is produced by a JTS of size s_i that is constructed by the Sharing JTS Algorithm.*

PROOF. Let Q be a query, P a profile and S the set of JTSs, such that, each JTT in $PRes(Q, P)$ can be produced by a JTS in S. S is divided into two sets S_1 and S_2, such that, $S_1 \cap S_2 = \emptyset$ and $S_1 \cup S_2 = S$. S_1 consists of all JTSs containing both the tuple sets $R_i^{\{w_r\}}$, $R_j^{\{w_r, w_t\}}$ for a selected keyword $w_r \in Q$, $\forall w_t \in W_{P_Q}$, and S_2 all the rest. With respect to Algorithm 3, JTSs of S_2 are constructed through the lines 3-5, 11-17 and 33-42, while JTSs of S_1 are constructed through the lines 18-42. Therefore, in any case, every JTT in $PRes(Q, P)$, can be produced by a JTS constructed by the *Sharing JTS Algorithm*. □

4.3 Top-k Query Processing

In the previous section, we introduced the *Sharing JTS Algorithm* that efficiently constructs all JTSs for a query Q. Next, we focus on how to retrieve the top-k results for Q (see Definition 9). In general, we use the function $\mathcal{F}$ to determine the number of JTTs each level contributes to the result, thus calibrating preferential dominance, while the specific trees of the result are selected based on their relevance, coverage and diversity.

Relevance is tuned through the maximum size s of the JTSs constructed with regards to Algorithms 2 and 3, while coverage is ensured by selecting trees from each level i, so that, as many keywords as possible are represented in the final result. Concerning diversity, we have to identify the trees with the maximum pair-wise distances.

Given the set $\mathcal{Z} = \bigcup_i Z^i$ of m relevant JTTs, our goal is to produce a new set S, $S \subset \mathcal{Z}$, with the k most diverse JTTs, $k < m$, such that, Z^i contributes $\mathcal{F}(i)$ trees. The problem of selecting the k items having the maximum average pair-wise distance out of m items is similar to the *p-dispersion-sum* problem. This problem as well as other variations of the general p-dispersion problem (i.e. select p out of m points, so that, the minimum distance between any two pairs is maximized) have been studied in operations research and are in general known to be NP-hard [13].

Algorithm 3 Sharing JTS Algorithm

Input: A profile P, a set of queries KQ of the form $Q_t = Q \cup \{w_t\}$, $\forall w_t \in W_{P_Q}$, a schema graph $\mathcal{G}_U$ and a size s.
Output: A list JTList of JTSs with size up to s for the queries in KQ.

1: **begin**
2: $Queue_1$, Q': queues of JTSs;
3: JTS^Q, $JTSList$: empty lists;
4: Compute the tuple sets R_i^X with regards to Q;
5: Select a keyword $w_r \in Q$;
6: Construct and insert to JTS^Q the JTSs of Q; /* as steps 9-27 of the Baseline JTS Algorithm */
7: l = 1;
8: **while** unmarked keywords exist in W_{P_Q} **do**
9: Compute the set of keywords $win_{P_Q}(l)$;
10: **for** each $Q_t \in KQ$, such that, $w_t \in win_{P_Q}(l)$ **do**
11: Mark w_t;
12: **for** each R_i^X, $1 \leq i \leq n$, computed for Q **do**
13: Construct the tuple sets R_i^X and $R_i^{X \cup \{w_t\}}$ for Q_t;
14: **end for**
15: **for** each JTS in JTS^Q **do**
16: Construct all combinations of trees of tuple sets by replacing the tuple sets of Q with the relative tuple sets of Q_t;
17: Insert those JTSs into $Queue_1$;
18: **end for**
19: **for** each $R_i^{\{w_r\}}$, $1 \leq i \leq n$, computed for Q_t **do**
20: Insert $R_i^{\{w_r\}}$ into $Queue_2$;
21: **end for**
22: **while** $Queue_2 \neq \emptyset$ **do**
23: Remove the head B from $Queue_2$;
24: **for** each R_i^X, such that, there is an R_j^Y in B and R_i is adjacent to R_j in $\mathcal{G}_U$ **do**
25: **if** $X = \{w_r, w_t\}$ AND size of $B < s - 1$ **then**
26: Expand B to include R_i^X;
27: Insert the updated B into $Queue_1$;
28: **else if** $X = \{\}$ AND size of $B < s - 1$ **then**
29: Expand B to include R_i^X;
30: Insert the updated B into $Queue_2$;
31: **end if**
32: **end for**
33: **end while**
34: **while** $Queue_1 \neq \emptyset$ **do**
35: Remove the head B from $Queue_1$;
36: **if** T satisfies the pruning rule **then**
37: Ignore B;
38: **else if** $keys(B) = Q \bigcup \{w_t\}$ **then**
39: Insert B into $JTList$;
40: **else**
41: /* as steps 21-26 of the Baseline JTS Algorithm */
42: **end if**
43: **end while**
44: **end for**
45: l++;
46: **end while**
47: **return** $JTlist$;
48: **end**

A brute-force method to locate the k most diverse JTTs of $\mathcal{Z} = \bigcup_i Z^i$, $|\mathcal{Z}| = m$, is to first produce all $\binom{m}{k}$ possible combinations of trees and then pick the one with the maximum set diversity out of those that satisfy the constraints of Definition 9. The complexity of this process is exponential and therefore, the computational cost is too high even for low values of m and k. A number of lower-complexity heuristics have been proposed to locate subsets of elements (e.g. in [13]). In this paper, we use the following variation: we construct a diverse subset of JTTs based on the tree-set distance.

DEFINITION 10 (TREE-SET DISTANCE). *Given a JTT T and a set of JTTs $S = \{T_1, \ldots, T_z\}$, the tree-set distance between T and S is:*

Algorithm 4 Top-k JTTs Algorithm

Input: The sets of keywords $win_{P_Q}(1), \ldots, win_{P_Q}(l)$ and the sets of JTTs $Z^1, \ldots, Z^l, Z^{l+1}$.
Output: The set S of the top-k JTTs.

1: **begin**
2: $S = \emptyset$;
3: **for** $i = 1$; $i <= l$; i++ **do**
4: **for** each $j \in win_{P_Q}(i)$ **do**
5: $counter(i, j) = \frac{\mathcal{F}(i)}{|win_{P_Q}(i)|}$;
6: **end for**
7: **end for**
8: Find the trees $T_1, T_2 \in Z^1$ with the maximum distance;
9: $S = S \cup T_1$;
10: $S = S \cup T_2$;
11: **for** $i = 1$; $i <= l + 1$; i++ **do**
12: **for** $j = 0$; $j < \mathcal{F}(i)$; j++ **do**
13: Find the tree $T \in Z^i \backslash S$ with the maximum $dist(T, S)$;
14: $S = S \cup T$;
15: **if** $i < l + 1$ **then**
16: Find the keyword w that T was computed for;
17: $counter(i, w) = counter(i, w) - 1$;
18: **if** $counter(i, w) == 0$ **then**
19: Remove from Z^i all JTTs computed for w;
20: **end if**
21: **end if**
22: **end for**
23: **end for**
24: **end**

$$dist(T, S) = \min_{1 \leq i \leq z} d(T, T_i).$$

Initially, we consider an empty set S. We first add to S the two furthest apart elements of Z^1. Then, we incrementally construct S by selecting trees of $Z^1 \backslash S$ based on their tree-set distance from the trees already in S. In particular, we compute the distances $dist(T_i, S)$, $\forall T_i \in Z^1 \backslash S$ and add to S the tree with the maximum corresponding distance. When $\frac{\mathcal{F}(1)}{|win_{P_Q}(1)|}$ trees have been added to S for a keyword in $win_{P_Q}(1)$, we exclude JTTs computed for that keyword from Z^1. After $\mathcal{F}(1)$ trees have been selected from Z^1, we proceed by selecting trees from $Z^2 \backslash S$ until another $\mathcal{F}(2)$ trees have been added to S and so on.

We can further reduce the number of performed operations based on the observation that after the insertion of a tree T to S, the distances of all other trees that have not yet entered the diverse results from S', $S' = S \cup \{T\}$, are affected only by the presence of T. This leads us to the following proposition:

PROPOSITION 2. *Given a JTT T_i and two sets of JTTs S and S', $S' = S \cup \{T_i\}$, it holds that:*

$$dist(T_j, S') = \min\{dist(T_j, S), d(T_j, T_i)\}.$$

The above process is shown in Algorithm 4. Observe that, threshold-based top-k algorithms (e.g. [14]) cannot be applied to construct diverse subsets of JTTs, since the $k - 1$ most diverse trees of $\mathcal{Z}$ are not necessarily a subset of its k most diverse ones.

5. EXTENSIONS

In this section, we consider extending the preference model by relaxing its context part and allowing more keywords in its choice part. We also discuss a simple approach for deriving preferences.

5.1 Relaxing Context

For a profile P and a query Q, the associated set of preferences P_Q may be empty, that is, there may be no preferences for Q. In this case, we can use for personalization those preferences whose context is more general than Q, i.e. their context is a subset of Q.

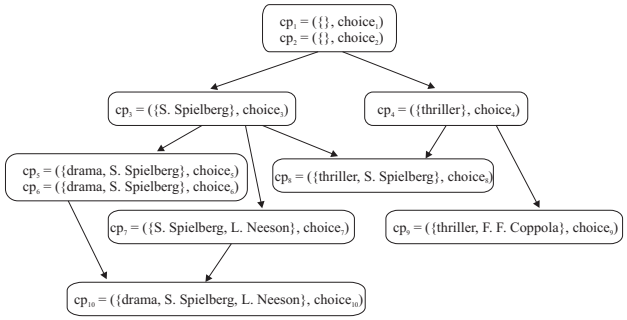


Figure 3: Context lattice of preferences.

DEFINITION 11 (RELAXED CONTEXT). *Given a query Q and a profile P, a set $C \subset Q$ is a relaxed context for Q in P, if and only if, (i) $\exists$ $(C, choice) \in P$ and (ii) $\nexists$ $(C', choice') \in P$, such that, $C' \subset Q$ and $C \subset C'$.*

Given a profile P, the *relaxed preferential result* of a query Q is the set of all JTTs that are both total and minimal for at least one of the queries $Q \bigcup \{w_i\}$, $w_i \in W_{P_C}$, where C is a relaxed context for Q. That is, we do not relax the original query Q, but instead, we just use the choice keywords of a relaxed context for Q.

To depict the subset relation among contexts, a lattice representation can be used. Any context-free preference is placed on the top of the lattice. An example is shown in Figure 3. For instance, given the preferences of Figure 3 and the query Q = {*thriller, F. F. Coppola, R. DeNiro*}, since there is no preference with context equal to Q, the choice keywords of preference cp_9, whose context is a relaxed context for Q, will be used. Note that the context of cp_4 is also more general than Q, but it is not a relaxed context for Q because the context of cp_9 is more specific.

If there is no preference more specific to Q, we finally select the context-free preferences, if any. Finally, note that there may be more than one relaxed context for Q. For instance, for the query Q = {*thriller, S. Spielberg, L. Neeson*}, both {*thriller, S. Spielberg*} and {*S. Spielberg, L. Neeson*} are relaxed contexts. In this case, we can use either of them. We could also use more than one relaxed context but this raises semantic issues with regards to the composition of the associated orders between their choice keywords, if they are conflicting, which is an issue beyond the scope of this work.

5.2 Multi-Keyword Choices

Our model of preferences supports choices between two keywords. One may think of more complex preferences of the form $(C, choice)$, where $C \subseteq W$ and $choice = (w_{l_1} \wedge \ldots \wedge w_{l_x}) \succ (w_{r_1} \wedge \ldots \wedge w_{r_y})$, w_{l_j}, w_{r_z}, $1 \leq j \leq x$, $1 \leq z \leq y$, $\in W$. We shall refer to such preferences as *composite contextual keyword preferences*. As an example, consider the preference *ccp* = ({*comedy, W. Allen*}, (*E. Norton* $\wedge$ *D. Barrymore*) $\succ$ (*B. Crystal* $\wedge$ *D. Moore*)). The meaning of *ccp* is that in the case of *comedy* movies and *W. Allen*, those movies that are related to both *E. Norton* and *D. Barrymore* are preferred over those that are related to both *B. Crystal* and *D. Moore*. Choices can be constructed arbitrarily, in the sense that each choice can have any number of keywords and different number of keywords can be used for the left and the right part, i.e. it may hold that $x \neq y$.

Supporting composite preferences of this form is straightforward. In this case, the preferential result of a query Q is the set of all JTTs that are both total and minimal for at least one of the queries $Q \bigcup W_i$, where W_i is now a set of keywords that appear in one of the parts of a choice for Q. The algorithms of Sections 4.2, 4.3 can be applied without any modifications. However, to speed up query processing when preferences with composite choices are used, we can further exploit the main idea of the *Sharing JTS Algorithm*. In particular, consider a query Q and two sets of keywords W_1, W_2 that appear in the choices of the relevant to Q preferences. During construction, the JTSs of $Q \bigcup W_1$ and $Q \bigcup W_2$ will be computed. Assuming that $W_1 \bigcap W_2 \neq \{\}$, we could first compute the JTSs of $Q \bigcup (W_1 \bigcap W_2)$ and then use them to find the JTSs of $Q \bigcup W_1$ and $Q \bigcup W_2$, instead of computing them from scratch.

5.3 Profile Generation

User preferences can either be explicitly provided by the user or be automatically constructed based on the previous user interactions or other available information. Although the focus of this paper is on how to exploit already constructed profiles to personalize keyword database search, we also discuss here a method for potentially inferring contextual keyword preferences in the absence of user input.

Assume that we maintain a log H of the keyword queries submitted to the database. To allow multiple occurrences of the same query Q in H, let us assume that each submitted query is preceded in the log by a unique identifier, that is, H is a set of entries of the form (id, Q) for each submitted query, where id is a unique identifier and Q the content of the query, that is, its keywords. For instance, H ={(id_1, {*thriller, G. Oldman*}), (id_2, {*drama, S. Spielberg*}), (id_3, {*drama, Q. Tarantino*}), (id_4, {*drama, 1993, S. Spielberg*}), (id_5, {*comedy, W. Allen*}), (id_6, {*drama, S. Spielberg*})} is a log of six queries, where, for example, the query {*drama, S. Spielberg*} was submitted twice.

Let W' be a set of keywords, $W' \subseteq W$. We use $freq(W')$ to denote the number of queries in H in which W' appears: $freq(W')$ = $|\{(id, Q) \in H$, such that, $W' \subseteq Q\}|$. For instance, in the example H, $freq(\{drama\})$ = 4. Our underlying assumption is that high popularity of a set W' of keywords, i.e. a large $freq(W')$ value, implies a preference on W', which is an assumption commonly made by many preference learning algorithms [17, 4]. More precisely, when a keyword w_i appears together with a set of other keywords W', i.e. in the same query with them, more frequently than a keyword w_j does, this is considered as an indication that w_i is preferred over w_j in the context of W'. Thus, we create a contextual preference $(W', w_i \succ w_j)$, for w_i, $w_j \notin W'$, if $freq(W' \bigcup \{w_i\}) - freq(W' \bigcup \{w_j\}) \geq minf \times |H|$, where $minf < 1$ is a positive constant that tunes the strength of the preferences. For instance, for the example H and $minf$ = 0.30, we infer the contextual keyword preference ({*drama*}, *S. Spielberg* $\succ$ *Q. Tarantino*).

Note that the above rule, for context-free preferences, i.e. for $W' = \{\}$, gives us $w_i \succ w_j$ if $freq(\{w_i\}) - freq(\{w_j\}) \geq minf \times |H|$, which simply gives priority to popular keywords over less popular ones. Recall that through context relaxation, context-free preferences will be applied when nothing more specific exists. This means that, for instance, for a query whose keywords have not appear in any of the queries in H, we can use such context-free preferences to personalize it.

6. EVALUATION

To evaluate the efficiency and effectiveness of our approach, we conducted a number of experiments, using both real and synthetic datasets: (i) the MOVIES database [2] (Figure 4) and (ii) the TPC-H database [3]. The MOVIES dataset consists of nearly 11500 movies, 6800 actors, 3300 directors, 175 studios and more than 45000 play tuples. For the TPC-H database, to experiment with the distribution of each keyword's appearance, we do not use its actual dataset but rather only its schema and generate data using the following procedure (that was also used in [19]). Each keyword appears in a relation R_i of the database with a probability equal to

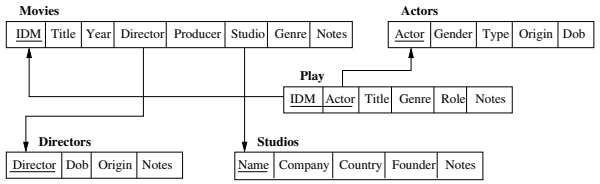


Figure 4: MOVIES schema.

$\frac{\log(\text{size}(R_i))}{x \cdot \log(y)}$, where size$(R_i)$ is the cardinality of R_i and y is the cardinality of the largest relation in the database. The lower the value of x, the higher the probability for a keyword to appear in a relationship.

We run our performance experiments for (i) queries of a different size $|q|$, (ii) profiles with a different number of preferences and thus, a different number $|w|$ of relevant choice keywords, (iii) various maximum sizes s for the computed JTTs and (iv) different keyword selectivities. We use MySQL 5.0 to store our data. Our system is implemented in JDK 1.5 and connects to the DBMS through JDBC. We use an Intel Pentium D 3.0GHz PC with 1GB of RAM. The profiles and queries used in our experiments along with the source code and datasets are available for download [1].

6.1 Performance Evaluation

In our performance evaluation study, we focus on (i) highlighting the efficiency of the *Sharing JTS Algorithm*, (ii) demonstrating the effectiveness of the *Top-k JTTs Algorithm* and (iii) assessing the overhead of query personalization as well as the reduction in the result size achieved.

6.1.1 Sharing vs. Baseline JTS Algorithm

To illustrate the efficiency of the *Sharing JTS Algorithm* versus the *Baseline* alternative, we measure the execution time and the total number of join operations performed during the phase of JTSs expansion.

Figures 6a, 6b report the execution time and total number of join operations, for the TPC-H database, for $|w| = 10$ when $|q|$ and s vary, while Figures 6c, 6d show the values of the corresponding measures for $|q| = 3$ and varying $|w|$, s. In all cases, we consider that $x = 10$, which means that the probability of a keyword appearing in the largest relation ($LINEITEM$) is 10%, while for the smallest relation ($REGION$), this probability is around 1%. The *Sharing JTS algorithm* is more efficient, performing only a small fraction of the join operations performed by the *Baseline* one, thus, also requiring much less time. As s increases, the reduction becomes more evident, since the larger this size is, the more the computational steps that are shared. For example, in Figure 6a, when $s = 5$, the *Sharing JTS Algorithm* requires only 2.5%-10.5% of the time required by the *Baseline JTS Algorithm*. Observe that, while the number of joins for the *Baseline JTS Algorithm* increases along with $|q|$, it decreases for the *Sharing JTS Algorithm* (Figure 6b). This happens because for a larger value of $|q|$, the trees of the preferential result share larger common sub-trees, therefore the *Sharing JTS Algorithm* performs fewer expansions.

To study the impact of keyword selectivity, we also run a set of experiments for $x = 10, 8, 6$ and $s = 3, 4, 5$ for constant values of $|q|$ and $|w|$ ($|q| = 3$ and $|w| = 10$). The results are shown in Table 1. For lower values of x, i.e. higher keyword selectivity, both the execution time and join operations increase for both algorithms, since more results exist. In all cases though, the *Sharing JTS Algorithm* outperforms the *Baseline* one. For example, for $x = 8$, the reduction in execution time is around 90%, while the join operations are reduced by 80%.

Similar observations can be made for the MOVIES database. In this case, we manually picked keywords with various selectivities, trying to construct queries and profiles that lead to results of different sizes and relevance. The *Sharing JTS Algorithm* requires around 10% of the time required by the *Baseline JTS Algorithm*, while the reduction of join operations during the expansion phase depends on $|q|$ and varies from 90% to 50%. The corresponding results are shown in Figure 7.

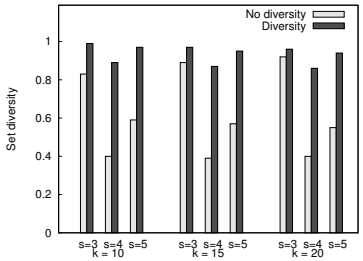


Figure 5: MOVIES dataset: Set diversity of first-level results.

Table 1: TPC-H dataset: Varying keyword selectivity.

x	*s*	Time (msec)		Number of joins	
		Baseline	Sharing	Baseline	Sharing
10	3	10.8	2.38	73.7	14.59
	4	68.8	5.31	451.45	87.65
	5	618.99	32.5	3402.5	666.12
8	3	12.19	2.5	87.44	17.29
	4	84.28	5.8	524.1	108.7
	5	701.20	38.19	3877.93	752.75
6	3	14.81	2.74	107.68	22.08
	4	91.64	6.67	605.91	126.22
	5	805.91	50.34	4277.75	950.37

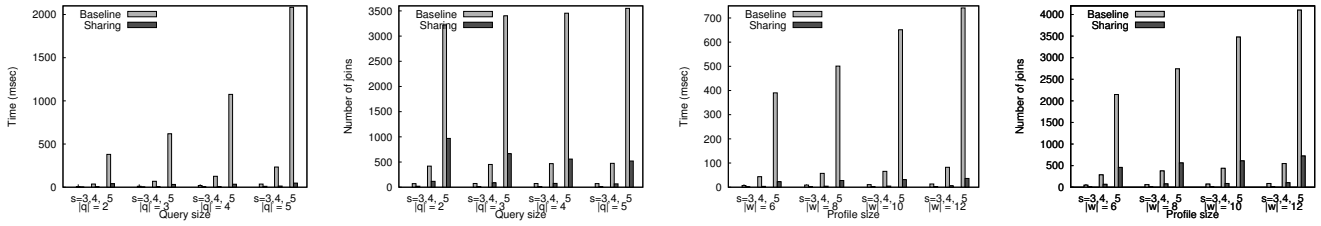
6.1.2 Top-k JTTs Algorithm

Our *Top-k JTTs Algorithm* combines four metrics in determining the top-k results for a query q, namely, preferential dominance, degree of relevance, coverage and diversity. To compute the overall result, we use a number of heuristics that guide the order of generation of the JTTs. We first evaluate the performance of our basic heuristics and then show their effectiveness.

First, we evaluate the performance of our underlying diversification heuristic by comparing it against the brute-force algorithm both in terms of the quality of produced results as well as the time. The complexity of all methods depends on the number m of candidate trees to choose from and on the required number k of trees to select. We experiment with a number of different values for m and k. However, the exponential complexity of the brute-force algorithm prevents us from using large values for these two parameters. Therefore, we limit our study to $m = 10, 20, 30$ and $k = 4, 8, 12, 16, 20$. In Table 2, we show the results for the brute-

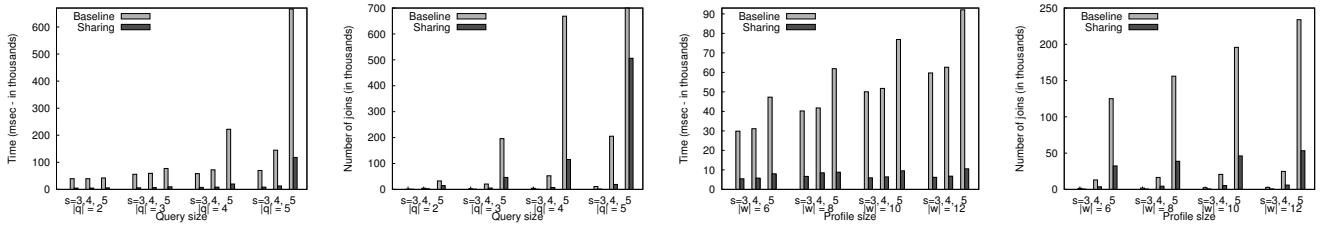
Table 2: Brute-force vs. Heuristic diversification.

m	*k*	Brute-force		Heuristic	
		Set diversity	Time (msec)	Set diversity	Time (msec)
10	4	0.98	33	0.97	7
	8	0.92	38	0.92	11
20	4	0.99	623	0.97	16
	8	0.94	71194	0.93	21
	12	0.86	171315	0.86	30
	16	0.81	11730	0.80	43
30	4	1.00	3190	0.99	21
	8	0.98	3041457	0.98	30
	12	0.96	105035021	0.95	43
	16	0.94	300561487	0.93	61
	20	0.91	104214544	0.90	79



(a) (b) (c) (d)

Figure 6: TPC-H dataset: Total time for (a) a fixed profile and (c) a fixed query and total number of join operations for (b) a fixed profile and (d) a fixed query.



(a) (b) (c) (d)

Figure 7: MOVIES dataset: Total time for (a) a fixed profile and (c) a fixed query and total number of join operations for (b) a fixed profile and (d) a fixed query.

force method and our heuristic. We observe that the brute-force method consumes much more time than the heuristic, while the set diversity of the produced results is similar (the difference is less than 1%). Applying our diversity heuristic improves the set diversity of results, even when choosing trees of tuples only among those computed for the choice keywords of the same winnow level (Figure 5).

As discussed, we can tune the trade-off between dominance and relevance through the function $\mathcal{F}$. To demonstrate this, we use the function $\mathcal{F}(i) = k \cdot \frac{L-(i-1)}{\sum_i (L-(i-1))}$ for various values of L, where L is the lowest winnow level from which results are retrieved. For example, when $L = 1$, only results corresponding to the choice keywords of the first winnow level are returned. We run the following experiments for $|q| = 1$, $|w| = 10$ and $s = 4$. In Figures 8a, 8b, we use a profile leading to five winnow levels. We also consider a sixth level containing the query results when no preferences are used. We show the average normalized dominance and relevance respectively, for $L = 1, 2, \ldots, 6$. Given a set S of JTTs, the average dominance is the mean $dorder$ and $iorder$ of the trees in $PRes(Q, P)$ and $Res(Q)$ respectively, where for those JTTs in $Res(Q)$ that are not the projection of any JTT in $PRes(Q, P)$, we use $iorder$ = 6 (that is, the maximum winnow level plus 1) as opposed to ∞. As L increases, the average dominance decreases because less preferable choice keywords are also employed, while the average relevance increases, since highly relevant JTTs from the lower levels enter the top-k results.

Coverage is also very important, especially in the case of skewed selectivity among the choice keywords. For example, if the combination of the query keywords and some top-level choice keyword is very popular, then, without coverage, the JTTs computed for that choice keyword would dominate the result. Figure 8c shows the average coverage for two profiles, when our coverage heuristic is employed or not, for $L = 5$. The first profile (*Pr. A*) contains keywords with similar selectivities, while the second one (*Pr. B*) contains keywords of different popularity, i.e. some keywords produce more results than others. Coverage is greatly improved in both cases when the heuristic is applied, since more keywords from all winnow levels contribute to the result. The improvement is more evident for *Pr. B*, as expected.

In general, high coverage ensures that results reaching the users represent most of their interests. However, this does not necessarily mean that those results are not similar with each other. Next, to demonstrate how selecting results based on our diversity heuristic can produce even more satisfying results, we execute a number of queries and present here a characteristic example. For the query {*drama*} and a winnow level containing the keywords *Greek* and *Italian*, four results should be selected according to $\mathcal{F}$. For simplicity, we use only the JTTs computed for the joining trees of tuple sets $Movies^{\{Drama\}} - Directors^{\{Greek\}}$ and $Movies^{\{Drama\}} - Directors^{\{Italian\}}$. When only coverage is applied, the results are:

(i) (Tan21, Eternity and a Day, 1998, Th. Angelopoulos, **Drama**) − (Th. Angelopoulos, 1935, **Greek**)

(ii) (FF50, Intervista, 1992, F. Fellini, **Drama**) − (F. Fellini, 1920, **Italian**)

(iii) (Tan12, Landscape in Fog, 1988, Th. Angelopoulos, **Drama**) − (Th. Angelopoulos, 1935, **Greek**)

(iv) (GT01, Cinema Paradiso, 1989, G. Tornatore, **Drama**) − (G. Tornatore, 1955, **Italian**)

When diversity is also considered, the third of these results is replaced by (PvG02, Brides, 2004, P. Voulgaris, **Drama**) − (P. Voulgaris, 1940, **Greek**). Coverage remains the same, however, with diversity, one more director can be found in the results.

6.1.3 Result Pruning and Time Overhead

Finally, we study the overall impact of query personalization in keyword search in terms of the number of returned results and the corresponding time overhead. Both of these measures depend on how frequently the relative to the query choice keywords appear in the database. Therefore, we experiment with profiles with different selectivities. As *profile selectivity* we define the normalized sum of the number of appearances of each choice keyword in the database. We use profiles with $|w|$ = 6, 8, 10, 12 and study a query with $|q| = 2$: (i) when no preferences are applied or a profile with (ii) small and (iii) large selectivity is used.

In Figures 9a and 9b, we measure the total number of the constructed JTTs, i.e. not just the top-k ones, for s = 3, 4 respectively. In general, query personalization results in high pruning. For $s = 3$, the use of the profile with large selectivity prunes more than 85% of the initial results, while for the profile with small selectivity the pruning is more than 95%. The respective percentages for $s = 4$ are 33% and 74%. In Figures 9c and 9d, we measure the time to generate the joining trees of tuple sets required to retrieve the final results. When the profile with large selectivity is applied, the time overhead is 24% for $s = 3$ and 35% for $s = 4$ on average. For the profile with small selectivity, the corresponding percentages are 22% and 32% on average.

6.2 Usability Evaluation

The goal of our usability study is to demonstrate the effectiveness of using preferences. In particular, the objective is to show that for a reasonable effort of specifying preferences, users get more satisfying results. To this end, we conducted an empirical evaluation of our approach using the MOVIES dataset, with 10 computer science students with a moderate interest in movies. Each of them provided a set of contextual keyword preferences including context-free ones. On average, there were five preferences related to each of the queries that were later submitted by each user. Users were asked to evaluate the quality of the top-10 JTTs retrieved. For characterizing the quality, we use two measures: (i) precision and (ii) degree of satisfaction. The first one captures the judgment of the users for each individual result. In particular, users marked each result with 1 or 0, indicating whether they considered that it should belong to the top-10 ones or not. The ratio of 1s corresponds to the precision of the top-10 results, namely *precision*(10). The second measure evaluates the perceived user satisfaction by the set of results as a whole. To assess this, users were asked to provide an overall degree of satisfaction (*dos*) in the range [1, 10] to indicate how interesting the overall result set seemed to them.

We compare the results of keyword queries when executing them: without using any of the preferences and with using the related contextual keyword preferences, first based only on dominance and relevance and then based on all four properties. Also, we consider using only context-free preferences as well as a case in which there is no preference with context equal to the query and so, relaxation is employed. We use $\mathcal{F}$ as in our performance experiments with L equal to the maximum winnow level for each user. Table 3 reports the average values of the quality measures (we omit the detailed per user scores due to space limitations). Our results indicate that, when no preferences are employed, both *precision* and *dos* are low. The use of context-free preferences improves both measures moderately, since such preferences capture only the generic interests of each user. Applying contextual keyword preferences improves quality considerably, even when preferences with relaxed context are employed. The most satisfying results are produced when all properties are taken into account. This demonstrates how important set-centric properties are when combined with dominance and relevance. Although our evaluation is preliminary, we believe that the results attained so far are promising.

Concerning user behavior, in general, most of our users defined preferences that resulted in short graphs of choices. Short graphs produce few winnow levels and consequently, many ties among the results with respect to preferential dominance. Using relevance, coverage and diversity led to resolving such ties. We also noticed that our users were often positively biased for movies they have heard about. This can be seen as an indication that exploiting previous queries to generate additional preferences based on popularity (as explained in Section 5.3) can prove very useful.

Table 3: Usability Evaluation.

	precision(10)	**dos**
No Preferences	0.09	1.9
Context-Free Keyword Preferences	0.21	2.7
Relaxed Context	0.87	7.4
Contextual Keyword Preferences		
Dominance-Relevance	0.89	7.9
Dominance-Relevance-Coverage-Diversity	0.94	8.7

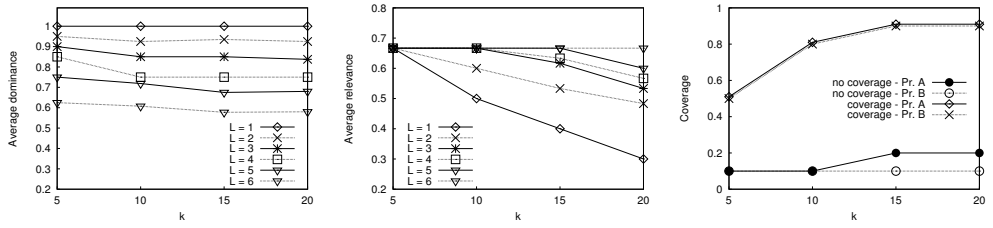
7. RELATED WORK

Keyword search in relational databases has been the focus of much current research. Schema-based approaches (e.g. [19, 6]) use the schema graph to generate join expressions and evaluate them to produce tuple trees. This is the approach we followed in this paper. Instance-based approaches (e.g. [9]) represent the database as a graph in which there is a node for each tuple. Results are provided directly by using a Steiner tree algorithm. Based on [9], several more complex approaches have been proposed (e.g. [16, 21]). There have also been proposals for providing ranked keyword retrieval, which include incorporating IR-style relevance ranking ([18, 26, 28]), authority-based ranking ([8]) and automated ranking based on workload and data statistics of query answers ([10]). Our approach is different, in that, we propose using preferences to personalize the results of keyword search. In this respect, précis queries ([29]) are the most relevant. Précis are keyword queries whose answer is a synthesis of results, containing tuples directly related to the given keywords and tuples implicitly related to them. While précis provides additional meaning to the results by adding structure, our goal is to use preferences for ranking results. Preferences are considered in [24] in the context of IR for document retrieval. The main difference is that, only the keywords that appear in the query are considered, whereas in our approach we expand the original query with choice keywords.

With regards to *preferences*, the research literature is extensive. There are two fundamental approaches for expressing preferences: a qualitative and a quantitative one. In the *qualitative approach* (e.g. [11, 22, 15]), preferences between items are specified directly, typically using binary preference relations. In the *quantitative approach* (e.g. [5, 23, 25, 7]), preferences are expressed indirectly by using scoring functions that assign numeric scores to items. We have followed a qualitative approach, since we think it is more natural for the user to express preferences among keywords directly. Using a quantitative approach is also feasible. Intuitively, the score of each keyword would correspond to its winnow level.

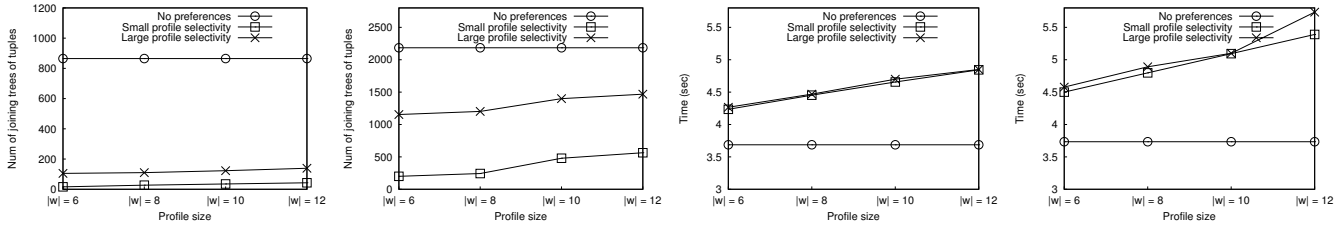
Previous contextual preference models, such as [31, 30], use the term *context* to refer to situational context, such as time and location. The most similar contextual model to ours is that in [4]. However, the work in [4] assumes knowledge of the schema and addresses a different problem: pre-computation of database rankings for representative contexts.

Recently, *diversity* has attracted considerable attention as a means for enhancing user satisfaction in recommender systems and web search (e.g. [36]). In terms of database queries, [33] considers diversifying the results over queries on a single database relation. The main idea is to build an appropriate B+-tree on the relation and explore a bounded number of tuples by using the tree to skip over similar tuples. A central difference with our approach is that our algorithms are built on top of the database engine. This may introduce some overheads, but it does not require any modifications of the database system. Besides algorithms for increasing diversity, a central issue is deriving an appropriate definition of diversity. In this paper, we used a Jaccard-based distance measure on the con-



(a) (b) (c)

Figure 8: (a) Average dominance, (b) average relevance and (c) coverage.



(a) (b) (c) (d)

Figure 9: Number of joining trees of tuples for (a) $s = 3$ and (b) $s = 4$ and time overhead for (c) $s = 3$ and (d) $s = 4$.

tent of the JTTs. Recently, [34] proposed a different view of diversity based on explanations which are based on past user ratings and proposed efficient algorithms to increase it. [27] considered the problem of selecting an appropriate set of features, so that, the differences among the results of structured queries are efficiently highlighted.

8. SUMMARY

The simplicity of keyword-based queries makes them a very popular method for searching. However, keyword-based search may return a large amount of matching data, often loosely related to the actual user intent. In this paper, we have proposed personalizing keyword database search by employing preferences. By extending query-relevance ranking with preferential ranking, users are expected to receive results that are more interesting to them. To further increase the quality of results, we have also suggested selecting k representative results that cover many user interests and exhibit small overlap. We have presented algorithms that extend current schema-based approaches for keyword search in relational databases to incorporate preference-based ranking and top-k representative selection.

9. REFERENCES

[1] PerK: Experimental Info (Source code and datasets). Available at http://www.cs.uoi.gr/∼kstef/PerK.

[2] Stanford Movies Dataset. Available at infolab.stanford.edu/pub/movies.

[3] TCP-H Dataset. Available at www.tcp.org.

[4] R. Agrawal, R. Rantzau, and E. Terzi. Context-sensitive ranking. In *SIGMOD*, 2006.

[5] R. Agrawal and E. L. Wimmers. A framework for expressing and combining preferences. In *SIGMOD*, 2000.

[6] S. Agrawal, S. Chaudhuri, and G. Das. Dbxplorer: A system for keyword-based search over relational databases. In *ICDE*, 2002.

[7] W.-T. Balke and U. Güntzer. Multi-objective query processing for database systems. In *VLDB*, 2004.

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

[9] G. Bhalotia, A. Hulgeri, C. Nakhe, S. Chakrabarti, and S. Sudarshan. Keyword searching and browsing in databases using banks. In *ICDE*, 2002.

[10] S. Chaudhuri, G. Das, V. Hristidis, and G. Weikum. Probabilistic information retrieval approach for ranking of database query results. *ACM Trans. Database Syst.*, 31(3), 2006.

[11] J. Chomicki. Preference formulas in relational queries. *ACM Trans. Database Syst.*, 28(4), 2003.

[12] W. W. Cohen, R. E. Schapire, and Y. Singer. Learning to order things. *J. Artif. Intell. Res. (JAIR)*, 10:243–270, 1999.

[13] E. Erkut, Y. Ülküsal, and O. Yeniçerioglu. A comparison of p-dispersion heuristics. *Computers & OR*, 21(10), 1994.

[14] R. Fagin, A. Lotem, and M. Naor. Optimal aggregation algorithms for middleware. In *PODS*, 2001.

[15] P. Georgiadis, I. Kapantaidakis, V. Christophides, E. M. Nguer, and N. Spyratos. Efficient rewriting algorithms for preference queries. In *ICDE*, 2008.

[16] H. He, H. Wang, J. Yang, and P. S. Yu. Blinks: ranked keyword searches on graphs. In *SIGMOD*, 2007.

[17] S. Holland, M. Ester, and W. Kießling. Preference mining: A novel approach on mining user preferences for personalized applications. In *PKDD*, 2003.

[18] V. Hristidis, L. Gravano, and Y. Papakonstantinou. Efficient ir-style keyword search over relational databases. In *VLDB*, 2003.

[19] V. Hristidis and Y. Papakonstantinou. Discover: Keyword search in relational databases. In *VLDB*, 2002.

[20] T. Joachims. Optimizing search engines using clickthrough data. In *KDD*, 2002.

[21] V. Kacholia, S. Pandit, S. Chakrabarti, S. Sudarshan, R. Desai, and H. Karambelkar. Bidirectional expansion for keyword search on graph databases. In *VLDB*, 2005.

[22] W. Kießling. Foundations of preferences in database systems. In *VLDB*, 2002.

[23] G. Koutrika and Y. Ioannidis. Personalized queries under a generalized preference model. In *ICDE*, 2005.

[24] A. Leubner and W. Kießling. Personalized keyword search with partial-order preferences. In *SBBD*, 2002.

[25] C. Li, K. C.-C. Chang, I. F. Ilyas, and S. Song. Ranksql: Query algebra and optimization for relational top-k queries. In *SIGMOD*, 2005.

[26] F. Liu, C. T. Yu, W. Meng, and A. Chowdhury. Effective keyword search in relational databases. In *SIGMOD*, 2006.

[27] Z. Liu, P. Sun, and Y. Chen. Structured search result differentiation. *PVLDB*, 2(1):313–324, 2009.

[28] Y. Luo, X. Lin, W. Wang, and X. Zhou. Spark: top-k keyword query in relational databases. In *SIGMOD*, 2007.

[29] A. Simitsis, G. Koutrika, and Y. Ioannidis. Précis: from unstructured keywords as queries to structured databases as answers. *VLDB J.*, 17(1), 2008.

[30] K. Stefanidis and E. Pitoura. Fast contextual preference scoring of database tuples. In *EDBT*, 2008.

[31] K. Stefanidis, E. Pitoura, and P. Vassiliadis. Adding context to preferences. In *ICDE*, 2007.

[32] R. Torlone and P. Ciaccia. Management of user preferences in data intensive applications. In *SEBD*, 2003.

[33] E. Vee, U. Srivastava, J. Shanmugasundaram, P. Bhat, and S. Amer-Yahia. Efficient computation of diverse query results. In *ICDE*, 2008.

[34] C. Yu, L. V. S. Lakshmanan, and S. Amer-Yahia. It takes variety to make a world: diversification in recommender systems. In *EDBT*, 2009.

[35] M. Zhang and N. Hurley. Avoiding monotony: improving the diversity of recommendation lists. In *RecSys*, 2008.

[36] C.-N. Ziegler, S. M. McNee, J. A. Konstan, and G. Lausen. Improving recommendation lists through topic diversification. In *WWW*, 2005.