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 database 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 \( Q = \{ \text{thriller}, \text{B. Pitt} \} \) 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: \((\{\text{thriller}\}, \text{G. Oldman} \succ \text{W. Allen})\) and \((\{\text{comedy}\}, \text{W. Allen} \succ \text{G. Oldman})\). 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]) 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 \( Q = \{ \text{thriller} \} \), 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 \( Q = \{ \text{thriller} \} \), we use the queries \( Q_1 = \{ \text{thriller}, \text{G. Oldman} \} \) and \( Q_2 = \{ \text{thriller}, \text{W. Allen} \} \). 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.
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 $R$ with $n$ relations $R_1, R_2, \ldots, R_n$. The schema graph $G_R$ is a directed graph capturing the foreign key relationships in the schema. $G_R$ 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 $G_U$.

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 foreign key dependencies [19, 6, 9].

**Definition 1 (Joining Tree of Tuples (JTT)).** Given an undirected schema graph $G_U$, 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 $(R_i, R_j) \in G_U$ and it holds that $(t_i \times t_j) \subseteq (R_i \times R_j)$.

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 $\text{Res}(Q)$ 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 $Q = \{\text{thriller}, \text{B. Pitt}\}$ consists of the JTTs: (i) $(m_2, \text{T. Gilliam})$ and $(a_2, \text{B. Pitt, male, 1963})$ and (ii) $(m_3, \text{Seven, thriller, 1996, D. Fincher})$.

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 $cp = (\{\text{thriller}, \text{B. Pitt}\}, \text{T. Gilliam} \succ \text{D. Fincher})$. 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 $T_a, T_b$ be two JTTs total for $Q$. We say that $T_a$ directly dominates $T_b$ under $P_Q$, $T_a \succ_{P_Q} T_b$, if and only if, $\exists w_i \in T_a, such that, \exists w_j \in T_b$ with $w_j \succ_{P_Q} w_i$ 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 $Q = \{w_1, w_2, w_3, w_4\}$, the choice keywords $w_1, w_2, w_3, w_4$ and the preferences $\{(w_1, w_2) \succ (w_1, w_3), (w_1, w_4) \succ (w_2, w_3), (w_4, w_3) \succ (w_4, w_2)\}$. Let $T_1, T_2$ be two JTTs in the result set of $Q$, where $T_1$ contains, among others, the keywords $w_2, w_1, w_3$ and $T_2$ the keywords $w_5$ and $w_2$. Then, based on Definition 4, although $T_1$ contains the keyword $w_2$ 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 $\succ_P$ 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 $T_1 \succ_P T_2$ and $T_2 \succ_P T_1$ hold. For instance, in the above example, assume $T_1$ with $w_2$ and $w_1$, and $T_2$ with $w_4$ and $w_2$. We consider such JTTs to be equally preferred. It is also possible that neither $T_1 \succ_P T_2$ nor $T_2 \succ_P T_1$ 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 $\{(\text{thriller}), F. F. \text{Coppola} \succ T. \text{Gilliam}\}$, for the query $Q = \{\text{thriller}\}$, the JTT $T_1 = \{\text{Dracula}, \text{thriller}, 1992, F. F. \text{Coppola}\}$ directly dominates the JTT $T_2 = \{\text{Twelve Monkeys}, \text{thriller}, 1996, T. \text{Gilliam}\}$. However, we cannot order results that may contain choice keywords indirectly through joins. For example, given the preference $\{(\text{thriller}), G. \text{Oldman} \succ B. \text{Pitt}\}$ and the same query $Q = \{\text{thriller}\}$, 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 $G. \text{Oldman}$, while $T_2$ is related to $B. \text{Pitt}$.

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, $T_j \in \text{Res}(Q)$. The set of the projected JTTs of $T_i$ for $Q$ is denoted by $\text{proj}_Q(T_i)$.

For example, assume the query $Q = \{\text{thriller}\}$. The JTT $(m_1, \text{Dracula}, \text{thriller}, 1992, F. F. \text{Coppola})$ is a projected JTT of $(m_1, \text{Dracula}, \text{thriller}, 1992, F. F. \text{Coppola}) \succ (m_1, a_1) \succ (a_1, G. \text{Oldman}, \text{male}, 1958)$ 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 (Preference Query Result):** Given a keyword query $Q$ and a profile $P$, the preference query result $\text{PRes}(Q, P)$ is the set of all JTTs that are both total and minimal for at least one of the queries $Q \cup \{w_i\}, w_i \in W_Q$.

Now, we can define indirect dominance as follows:

**Definition 6 (Indirect Preference 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 $P_Q$, $T_i \succ_P T_j$, if there is a JTT $T'_i \in \text{PRes}(Q, P)$, such that,

![Figure 2: The graph of choices $G_{P_{\text{thriller}, F. F. \text{Coppola}}}$.](image-url)
Algorithm 1 Multiple Level Winnow Algorithm

Input: A graph of choices $G_{PC} = (V_G, E_G)$.

Output: The sets $\text{win}_P(l)$ for the levels $l$.

1: begin
2: \text{winnow\_result} = empty list;
3: $l = 1$;
4: while $V_G$ not empty do
5: for all $w_i \in V_G$ with no incoming edges in $E_G$ do
6: \text{win}_P(l) = \text{win}_P(l) \cup \{w_i\};
7: end for
8: Add \text{win}_P(l) to \text{winnow\_result};
9: $V_G = V_G - \text{win}_P(l)$;
10: for all edges $e = (w_i, w_j)$ with $w_i \in \text{win}_P(l)$ do
11: $E_G = E_G - e$;
12: end for
13: $l++$;
14: end while
15: return \text{winnow\_result};
16: end

Theorem 2. Let $S = \bigcup_i \text{projection}(T_i), \forall T_i \in \text{Res}(Q, P)$, and $T_i$ be a JTT, such that, $T_i \in \text{Res}(Q)$. Then, $\forall T_j \in S$, it holds that (i) $T_j \succ_T T_i$ and (ii) $(T_i \succ_T T_j)$. 

Proof. Since $T_i \in S$ there is no $T_j, T_i \in \text{projection}(T_j)$, such that, $T_j$ contains a choice keyword of $W_Q$. However, for every $T_j \in S$ there is at least one $T_j, T_i \in \text{projection}(T_j)$, 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 $\text{Res}(Q, P)$, which is not minimal but provides an explanation of why its projected tree in $\text{Res}(Q)$ was ordered this way. For instance, for the query $Q = \text{[thriller, } G. \text{ Oldman > B. Pitt]}$ and the database instance in Figure 1, we could either present to the user as top result the JTT $(m_1, Dracula, \text{thriller, 1992, F. F. Coppola}) - (m_1, a_1) - (a_1, G. \text{ Oldman, male, 1958})$ that belongs to $\text{Res}(Q, P)$ or its projected JTT $(m_1, Dracula, \text{thriller, 1992, F. F. Coppola})$ that belongs to $\text{Res}(Q)$.

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 $\text{order}(T, Q)$. 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 $\text{relevance}(T, Q)$.

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 $Q = \text{[thriller, F. F. Coppola]}$ that include $R. \text{ DeNiro and R. Williams}$ 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 $R. \text{ DeNiro}$ but also JTTs that cover the preference on $R. \text{ Williams}$ and perhaps other choices as well. To capture this requirement, we define the coverage of a set 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 $S = \{T_1, \ldots, T_n\}$ of JTTs that are total for $Q$, the coverage of $S$ for $Q$ and $P$ is defined as:

\[
\text{coverage}(S, Q, P) = \frac{\bigcup_j \text{keywords}(S_j) \cap W_Q}{|W_Q|},
\]

where $\text{keywords}(S_j)$ is the set of keywords in $S_j$. 

588
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:
\[d(T_i, T_j) = 1 - \frac{|A \cap B|}{|A| + |B|} \]
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:
\[
diversity(S) = \frac{\sum_{i=1}^{z} \sum_{j>i}^{z} d(T_i, T_j)}{(z-1)\frac{z}{2}}.
\]
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 $Z^r = \bigcup_{w_i \in win_{P_Q}(r)} Res_s(Q \cup \{w_i\})$. Also, let $Z^{l+1} = Res_s(Q) \setminus \bigcup_{l \in P \setminus \text{Res}(Q,P)} \text{project}(Q)(T_e)$. 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 $F(i)$, where $F$ is a monotonically decreasing function with $\sum_{i=1}^{l+1} F(i) = k$. Each $Z^i$ contributes $F(i)$ 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 $F$, 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 choices $\{Z^1, \ldots, Z^l, Z^{l+1}\}$ with $|Z^1| + \ldots + |Z^l| + |Z^{l+1}| = m$, the top-$k$ JTTs, $k < m$, is the set $S^*$ for which:
\[
S^* = \text{argmax } diversity(S),
\]
such that $Z^i$ contributes $F(i)$ JTTs to $S^*$, which, for $1 \leq i \leq l$, are uniformly distributed among the keywords of winnow level $i$ and $F$ is a monotonically decreasing function with $\sum_{i=1}^{l+1} F(i) = k$.

There are two basic tuning parameters: the function $F$ and the threshold $s$. Dominance, coverage and relevance depend on how quickly $F$ 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 $\text{win}_{P_Q}(1)$ by setting, for example, $F(1) = k$, and $F(i) = 0$, for $i > 1$. A low decrease rate of $F$ 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 $F$ 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 $Q = \{\text{thiller, } B. \text{ Pitt}\}$. The corresponding result consists of the JTTs: (i) $(m_{22}, \text{Twelve Monkeys, } \text{thiller, } 1996, \ T. \text{ Gilliam})$ – $(m_{22}, a_{22} - a_{22}, \ B. \text{ Pitt, male, } 1963)$ and (ii) $(m_{21}, \text{Seven, thiller, } 1996, D. \text{ Fincher})$ – $(m_{21}, a_{22} - a_{22}, (B. \text{ Pitt, male, } 1963)$. Each JTT corresponds to a tree at schema level. For example, both of the above trees correspond to the schema level tree $\text{Movies}(\text{thiller}) - \text{Play}^1 - \text{Actors}(\text{B. Pitt})$, where each $R^i$ 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 (JTTs).

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 JTTs 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 $R^i_s$ are computed, where $R^i_s = \{t \mid t \in R_i \land \forall w \in X, t \text{ contains } w \land w \notin Q, X, t \text{ does not contain } w_q\}$. After selecting a random query keyword $w_q$ of all tuple sets $R^i_s$ for which $w_q \in X$ are located. These are the initial JTTs 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. JTTs that contain all query keywords are returned, while JTTs of the form $R^i_s - R^i_s - R^i_s$, where an edge $R_i \rightarrow R_i$ 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 JTTs for the queries $Q \cup \{w_i\}, \forall w_i \in \text{win}_{P_Q}(l)$, start-
Algorithm 2 Baseline JTS Algorithm

Input: A query \( Q \), a profile \( P \), a schema graph \( G_U \) and a size \( s \).
Output: A list JTList of JTSs with size up to \( s \) for the queries \( Q \cup \{ w_i \} \), \( \forall w_i \in W_{PQ} \).

1: begin
2: \[ Q \text{Queue: queue of JTSs; } \]
3: \[ JTList: \text{empty list; } \]
4: \[ l = 1; \]
5: while unmarked keywords exist in \( W_{PQ} \) do
6: Compute the set of keywords \( win_{PQ}(l) \);
7: for each \( w_i \in win_{PQ}(l) \) do
8: Mark \( w_i \);
9: Compute the tuple sets \( R^X_i \) for \( Q \cup \{ w_i \} \);
10: Select a keyword \( w_i \in Q \cup \{ w_i \} \);
11: for each \( R^X_i \), \( 1 \leq i \leq n \), such that, \( w_i \in X \) do
12: Insert \( R^X_i \) 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) = \( Q \cup \{ w_i \} \) then
19: Insert \( B \) into JTList;
20: else
21: for each \( R^X_i \), such that, there is an \( R^Y_j \) in \( B \) and \( R_i \) is adjacent to \( R_j \) in \( G_U \) do
22: if \( (X = \{ \} \text{ OR } X - \text{keys}(B) \neq \emptyset) \text{ AND (size of } B < s) \) then
23: Expand \( B \) to include \( R^X_i \);
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

JTS Algorithm (Algorithm 3) constructs first the JTSs for \( Q \) using a selected keyword \( w_r \in Q \) based on the tuple sets \( R^X_i \) for \( Q \) (lines 3-5). Then, for each \( Q \), we recompute its tuple sets by partitioning each \( R^X_i \) for \( Q \) into two tuple sets for \( Q_i \): \( R^X_i \) that contains the tuples with only the keywords \( X \) and \( R^X_i \cup \{ w_r \} \) that contains the tuples with only the keywords \( X \cup \{ w_r \} \) (lines 11-13). Using the JTSs for \( Q \) and the tuple sets for \( Q_i \), we produce all combinations of trees of tuple sets (lines 14-17) that will be used next to construct the final JTSs for \( Q \). For example, given the JTS for \( Q \) \( R^X_i - R^Y_i \), we produce the following JTSs for \( Q_i \): \( R^X_i - R^Y_i \), \( R^X_i \cup \{ w_r \} - R^Y_i \), \( R^X_i - R^Y_i \cup \{ w_r \} \) and \( R^X_i \cup \{ w_r \} - R^Y_i \). Note that, such a JTS is constructed only if all of its tuples sets are non-empty. The JTSs that contain all keywords of \( Q \) 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 \( R^X_i - R^Y_i \), the procedure described above does not construct for \( Q_i \) JTSs of the form \( R^X_i \cup \{ w_r \} - R^Y_i \). The same also holds for the JTSs that connect \( R^X_i \cup \{ w_r \}, R^Y_i \) 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 \( S_1 \cup S_2 = S \). \( S \) consists of all JTSs containing both the tuple sets \( R^X_i \cup \{ w_r \}, R^X_i \cup \{ w_r \} \) for a selected keyword \( w_r \in Q \), \( \forall w_r \in W_{PQ} \), 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 \( F \) 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 tunned 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 \( l \), 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 \( Z = \bigcup Z^* \) of \( m \) relevant JTSs, our goal is to produce a new set \( S, S \subseteq Z \), with the \( k \) most diverse JTTs, \( k < m \), such that, \( Z^* \) contributes \( F(i) \) trees. The problem of selecting the \( k \) items having the maximum average pair-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 \( Q_l = Q \cup \{w_l\} \), \( \forall w_l \in W_{PQ} \), a schema graph \( G_Q \) and a size \( s \).

Output: A list \( JTList \) of JTSs with size up to \( s \) for the queries in \( KQ \).

1: begin
2: Queue1, \( Q^i \) : queues of JTSs;
3: \( JTSS^Q \), \( JTSList \) : empty lists;
4: Compute the tuple sets \( R_i^X \) with regards to \( Q \);
5: Select a keyword \( w_l \in Q \);
6: Construct and insert to \( JTSS^Q \) the JTSs of \( Q ; P^* \) as steps 29-27 of the Baseline JTS Algorithm */
7: \( l = 1; \)
8: while unmarked keywords exist in \( W_{PQ} \) do
9: Compute the set of keywords \( win_{PQ}(l) \);
10: for each \( Q_l \in KQ \), such that, \( w_l \in win_{PQ}(l) \) do
11: Mark \( w_l \);
12: for each \( R_i^X \), \( 1 \leq i \leq n \), computed for \( Q \) do
13: Construct the tuple sets \( R_i^X \) and \( R_i^X \cup \{w_l\} \) for \( Q_l \); end for
14: for each JTS in \( JTSS^Q \) do
15: Construct all combinations of trees of tuple sets by replacing the tuple sets of \( Q \) with the relative tuple sets of \( Q_l \); end for
16: Insert those JTSs into \( Queue1 \);
17: end for
18: for each \( R_{l}^R \), \( 1 \leq i \leq n \), computed for \( Q_l \) do
19: Insert \( R_i^R \) into \( Queue2 \);
20: end for
21: while \( Queue2 \) \( \neq \emptyset \) do
22: Remove the head \( B \) from \( Queue2 \);
23: for each \( R_i^X \) such that, there is an \( R_i^Y \) in \( B \) and \( R_i \) is adjacent to \( R_i \) in \( G_Q \) do
24: if \( X = \{w_l, w_l\} \) AND size of \( B < s - 1 \) then
25: Expand \( B \) to include \( R_i^Y \);
26: Insert the updated \( B \) into \( Queue1 \);
27: else if \( X = \{\} \) AND size of \( B < s - 1 \) then
28: Expand \( B \) to include \( R_i^Y \);
29: Insert the updated \( B \) into \( Queue2 \);
30: end if
31: end for
32: end while
33: \( \emptyset \) \( \neq \emptyset \) do
34: Remove the head \( B \) from \( Queue1 \);
35: if \( T \) satisfies the pruning rule then
36: Ignore \( B \);
37: else if \( keys(B) = Q \cup \{w_l\} \) then
38: Insert \( B \) into \( JTList \);
39: end if
40: else
41: \( P^* \) 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

Algorithm 4 Top-\( k \) JTTs Algorithm

Input: The sets of keywords \( win_{PQ}(1), \ldots, win_{PQ}(l) \) 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 \( i = 1; i \leq l; i++ \) do
4: for each \( j \in win_{PQ}(i) \) do
5: \( counter(i, j) = \frac{|F(i)|}{|win_{PQ}(i)|} \);
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 \( i = 1; i \leq l + 1; i++ \) do
12: for \( j = 0; j < F(i); j++ \) do
13: Find the tree \( T \in Z^j \) with the maximum \( \text{dist}(T, S) \);
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

\[ \text{dist}(T, S) = \min_{1 \leq i \leq l} d(T, T_i). \]

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 \( Z^1 \setminus S \) based on their tree-set distance from the trees already in \( S \). In particular, we compute the distances \( \text{dist}(T_i, S) \), \( \forall T_i \in Z^1 \setminus S \) and add to \( S \) the tree with the maximum corresponding distance. When \( \frac{|F(i)|}{|win_{PQ}(i)|} \) trees have been added to \( S \) for a keyword in \( win_{PQ}(1) \), we exclude JTTs computed for that keyword from \( Z^1 \). After \( F(1) \) trees have been selected from \( Z^1 \), we proceed by selecting trees from \( Z^2 \setminus S \) until another \( F(2) \) 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 \( S^1, S^2 = S \cup \{T\} \), 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 \( S^1, S^2 = S \cup \{T_i\} \), it holds that:

\[ \text{dist}(T_j, S^2) = \min\{\text{dist}(T_j, S), d(T_j, T_i)\}. \]

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 \( Z \) 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 \( Q \).
Given a query \( Q \) and a profile \( P \), a set \( C \subseteq Q \) is a relaxed context for \( Q \) in \( P \) if and only if, (i) \( \exists (C, \text{choice}) \in P \) and (ii) \( \not \exists (C', \text{choice}') \in P \), such that, \( C' \subseteq Q \) and \( C \subseteq C' \).

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 \( Q \cup \{w_i\}, w_i \in W_{P_C} \), 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 \( Q = \{\text{thriller}, S.~Spielberg\} \), 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 \( cp_9 \) 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 \( Q = \{\text{thriller}, S.~Spielberg, L.~Neeson\} \), both \( \{\text{thriller}, S.~Spielberg\} \) 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.

Our model of preferences supports choices between two keywords. One may think of more complex preferences of the form \((C, \text{choice})\), where \( C \subseteq W \) and \( \text{choice} = \{w_{i_1} \land \ldots \land w_{i_y}\} \supseteq (w_{j_1} \land \ldots \land w_{j_x}), 1 \leq j \leq x, 1 \leq z \leq y, z \in W \). We shall refer to such preferences as composite contextual keyword preferences. As an example, consider the preference \( ccep = \{\text{ comedy, W.~Allen}, (E.~Norton \land D.~Barrmore) \supseteq (B.~Crystal \land D.~Moore)\} \). The meaning of \( ccep \) is that in the case of comedy movies and W. Allen, those movies that are related to both E. Norton and D. Barrmore 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 \( Q \cup \{w_i\} \), where \( W_i \) is 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 \( Q \cup W_1 \) and \( Q \cup W_2 \) will be computed. Assuming that \( W_1 \cap W_2 \neq \emptyset \), we could first compute the JTSs of \( Q \cup W_1 \) and then use them to find the JTSs of \( Q \cup W_2 \), 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, \( H = \{(id_1, \{\text{thriller, G.~Oldman}\}), (id_2, \{\text{drama, S.~Spielberg}\}), (id_3, \{\text{drama, Q.~Tarantino}\}), (id_4, \{\text{drama, 1993, S.~Spielberg}\}), \ldots\) \) is a log of six queries, where, for example, the query \( \{\text{drama}, S.~Spielberg\} \) 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: \( freq(W') = |\{(id, Q) \in H, \text{ such that, } W' \subseteq Q\}| \). For instance, in the example \( H \), \( freq(\{\text{drama}\}) = 4 \). 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 \( freq(W' \cup \{w_i\}) - freq(W' \cup \{w_j\}) \geq minf \times |H| \), 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 \((\{\text{drama}\}, S.~Spielberg \succ Q.~Tarantino)\).

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
\[ \log(\text{size}(R_i)) - x \log(y), \]  
where \text{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 JTTs 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 (\textit{LINEITEM}) is 10\%, while for the smallest relation (\textit{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.

#### 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-force and the heuristic algorithms.
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 $F(i) = k \cdot \frac{L(i-1)}{L(L-1)}$ 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 experiment 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 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 $\text{order} = 6$ (that is, the maximum winnow level plus 1) as opposed to $\infty$. 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 $\{\text{drama}\}$ 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 $\text{Movies}^{(\text{Drama})} - \text{Directors}^{(\text{Greek})}$ and $\text{Movies}^{(\text{Drama})} - \text{Directors}^{(\text{Italian})}$. When only coverage is applied, the results are:

(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.

![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.](image_url)

![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.](image_url)
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 $F$ 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.

<table>
<thead>
<tr>
<th></th>
<th>precision(10)</th>
<th>dos</th>
</tr>
</thead>
<tbody>
<tr>
<td>No Preferences</td>
<td>0.09</td>
<td>1.9</td>
</tr>
<tr>
<td>Context-Free Keyword Preferences</td>
<td>0.21</td>
<td>2.7</td>
</tr>
<tr>
<td>Relaxed Context</td>
<td>0.87</td>
<td>7.4</td>
</tr>
<tr>
<td>Contextual Keyword Preferences</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dominance-Relevance</td>
<td>0.89</td>
<td>7.9</td>
</tr>
<tr>
<td>Dominance-Relevance-Coverage-Diversity</td>
<td>0.94</td>
<td>8.7</td>
</tr>
</tbody>
</table>

### 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-
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.uio.no/~kstefi/PerK.