Fill-In-The-Functions: Towards Establishing A Workload For Ranking in Web Databases

Aditya Telang, Sharma Chakravarthy, Chengkai Li

Technical Report CSE 2010
Fill-In-The-Functions: Towards Establishing A Workload For Ranking in Web Databases

Aditya Telang, Sharma Chakravarthy, Chengkai Li

Department of Computer Science and Engineering, University of Texas at Arlington
aditya.telang@mavs.uta.edu, sharma@cse.uta.edu, cli@cse.uta.edu

September 25, 2010

Abstract

The emergence of the deep Web databases have given a new connotation to the concept of ranking query results. Earlier approaches for ranking resorted to analyzing frequencies of database values and query logs or establishing user profiles. In contrast, an integrated approach, based on the notion of a similarity model, for supporting user- as well as query-dependent ranking has been recently proposed [28]. An important component of this ranking framework is a workload of ranking functions, where each function represents an individual user’s preferences towards the results of a specific query. At the time of answering a query for which no prior ranking function exists, the similarity model can ensure a good quality of ranking only if a ranking function for a very similar user-query pair exists in the workload. Thus, when there exists no function corresponding to a user asking a query, the framework must ensure that the workload contains a ranking function for at least one user-query pair similar to this pair.

In this work, we address this problem of establishing an appropriate workload of ranking functions to support user- and query-dependent ranking on Web databases. Toward this, we propose a novel metric, termed as Workload Goodness, that determines the appropriateness of a given set of user-query pairs in assisting the Similarity model. We then prove that the optimal solution to this problem is NP-complete, and propose two distinct algorithms based on greedy approaches for individually determining an appropriate set of user-query pairs. We discuss the effectiveness of our proposal analytically as well as experimentally over two real Web databases.

1 Introduction

The emergence of the deep Web [7] [9] has led to the proliferation of a large number of Web databases for a variety of applications (e.g., airline reservations, vehicle search, real estate scouting). Typically, these databases are searched by formulating queries on their schema attributes, and often, these queries yield too many results. Currently, Web databases assist users by displaying the query results in a sorted order on the values of a single attribute (e.g., Price, Mileage, etc.). However, most Web users are likely to consider multiple attributes and their respective values for selecting the results relevant to them, and these preferences vary with respect to users and their queries. For example, consider the following scenario in the context of Google Base’s [14] Vehicle database that comprises of a table with attributes Make, Price, Color, and so on.

Example-1: Two users – a company executive (U1) and a student (U2), seek answers to the same query, Q1: “Make = Honda AND Location = Dallas, TX”. Intuitively, U1 would search for new vehicles, maybe with specific color choices (e.g., only red colored vehicles), and hence would prefer vehicles with “Condition = New AND Color = Red” to be ranked and displayed higher than the others. In contrast, U2 would most likely search for old vehicles priced under a specific amount (e.g., “Price < 5,000$”); hence, for U2, the resulting vehicles should be ordered such that the ones with “Condition = Old AND Price < 5,000$” are displayed higher than the others.
Let us assume that \( U_2 \) now moves to Mountain View, CA as an intern with Google and asks a different query, \( Q_2 \): “Make = Pontiac AND Location = Mountain View, CA”. We can presume (since he has procured an internship) that he may be willing to pay a slightly higher price for a lesser mileage vehicle (e.g., “Mileage < 100,000”), and hence would prefer vehicles with “Condition = Old AND Mileage < 100,000” to be ranked higher than others.

The above example illustrates that different Web users may have significantly different ranking preferences over the results of the same query. It further emphasizes that the same user may display varying ranking preferences for different queries (at different points in time). Thus, it is evident that in the setting of Web databases, where a large set of queries given by varied classes of users is involved, the corresponding results should be ranked in a user- and query-dependent manner.

In order to support such an integrated ranking scheme over Web databases, ranking preferences/functions over every possible user-query pair need to be acquired apriori (in order to rank the results at query time) – an impossible task. Extensions to SQL [25] [30] [22] [24] [31] have allowed manual specification of ranking preferences (or functions) at query time; however, this approach is difficult for most Web users. Although automated ranking of query results has been extensively researched in the domain of traditional and Web databases, current techniques [2] [11] [26] [32] only perform query-dependent ranking and do not differentiate between users. In contrast, techniques proposed for user-dependent ranking on Web databases do not differentiate between queries, and derive ranking functions for individual users by either building extensive user profiles [20] [21] or requiring users to order data tuples [1] [17].

To the best of our knowledge, the only framework that supports both, user- and query-dependent ranking, for ordering Web database query results has been proposed in [28]. This framework advances a similarity model based on the assumptions that: i) for the results of a given query, similar users display comparable ranking preferences, and ii) a user displays corresponding ranking preferences over results of similar queries.

An important component of this framework is a workload of ranking functions\(^1\) that is used by the similarity model for performing the subsequent ranking. Table 1 shows an instance of such a workload, represented as a matrix (\( W \)) of users and queries\(^2\). Each cell in this matrix (e.g., \( W[x,y] \)) represents a ranking function (e.g., \( F_{x,y} \)) derived for a distinct user-query pair (e.g., \( U_x \) and \( Q_y \)).

At the time of answering a query (say \( Q_j \)) asked by a user (\( U_i \)), if no ranking function (\( F_{i,j} \)) exists in this matrix, the similarity model, ideally, tries to obtain the ranking function deduced for either – i) the most similar user (to \( U_i \)) and the query (\( Q_j \)), or ii) the user (\( U_i \)) and the most similar query (to \( Q_j \)), or iii) the most similar user-query pair (to \( U_i \) and \( Q_j \) respectively). For instance, let us presume that \( U_1 \) is the most similar user to \( U_i \), and \( Q_1 \) is the most similar query to \( Q_j \) for the matrix shown in Table 1. Then, the similarity model would ideally prefer the availability of the functions – \( F_{1,j} \), or \( F_{i,1} \), or \( F_{1,1} \). However, since obtaining such a dense workload of ranking functions is practically impossible in a real setting of Web databases, the similarity model adopts a relaxed approach and avails of a ranking function deduced for a user-query pair occurring in the set of top-\( K \) user-query pairs (the preliminary details of determining such a set of pairs are elaborated in Section 2) most similar to the pair (\( U_i \), \( Q_j \)). For the workload shown in Table 1, assuming that the pair of \( U_x \) and \( Q_y \) satisfies the top-\( K \) requirement of the similarity model, the function \( F_{x,y} \) will be used for ranking the results of \( Q_j \) for \( U_i \).

In the absence of a ranking function in the set of top-\( K \) user-query pairs, the similarity model will be forced to use a ranking function (e.g., \( F_{1,45} \)) corresponding to a user and/or query which may not be very similar (in this case, \( Q_{45} \) may not be very similar to \( Q_j \)) to the input pair, thus, affecting the final quality of ranking achieved by the function. Therefore, in order to achieve an acceptable quality of ranking, for at least one of the top-\( K \) pairs similar to the pair (\( U_i \), \( Q_j \)), a ranking function should exist in the workload.

\(^1\)Each ranking function belonging to this workload corresponds to a distinct user-query pair i.e., it is derived based on an individual user’s preferences towards the results of a specific query.

\(^2\)For Web databases, although such a workload matrix can be extremely large, it will be very sparse as obtaining preferences for large number of user-query pairs is practically difficult. Hence, we have purposely represented the sample workload as a sparse matrix to emphasize the importance of establishing an appropriate workload for use by the similarity model.

\( \text{Table 1: An instance of such a workload.} \)

<table>
<thead>
<tr>
<th>User (( U ))</th>
<th>Query (( Q ))</th>
</tr>
</thead>
<tbody>
<tr>
<td>( U_1 )</td>
<td>( Q_1 )</td>
</tr>
<tr>
<td>( U_2 )</td>
<td>( Q_2 )</td>
</tr>
</tbody>
</table>


Table 1: Sample Workload Matrix $W$

Consequently, given a workload matrix $W$ comprising of $M$ queries and $N$ users (i.e., $M \times N$ total user-query pairs), our goal is to determine a set $S$ of user-query pairs such that, for each user-query pair in $W$, there exists at least one user-query pair in $S$ that occurs in the list of top-$K$ pairs most similar to the former. We call this the **Workload Filling Problem**.

Based on the above discussion, it is evident that more the number of ranking functions in the workload, better will be the overall average quality of ranking. However, in the context of Web databases, obtaining large number of functions from various user-query pairs is difficult, given the time and effort a user needs to spend in providing preferences to the results of a query (from which a function is derived). Hence, the size of the set $S$ i.e., the number of functions in the workload will be much smaller than the total size of the workload. Determining a pragmatic size for $S$ for a Web database is beyond the scope of this paper. Therefore, we assume that the size of this set ($|S|$) is provided by the Web application. For the sake of completeness, we have experimented with different cardinalities for this set (in Section 5) and analyzed the corresponding effect on the overall ranking quality that can be achieved by the similarity model.

We prove that finding the optimal solution (i.e., finding the optimal set of user-query pairs to represent $S$) for the Workload Filling problem is NP-complete in nature (we generalize the problem as a decision version of the Dominating Set [15] problem), and propose an approximate solution for the same. The appropriateness, of the set of user-query pairs, obtained from this solution is formalized based on a novel metric of **Workload Goodness** ($G(W,S)$) proposed in this paper. The intent of this metric is – given two sets of user-query pairs, $S_1$ and $S_2$ of the same size, it determines which set maximizes the goodness of the workload. We further show that finding the ideal set of a specified number (given as $|S|$) of user-query pairs is intractable and leads to a combinatorial explosion. Hence, we propose two greedy approaches that individually find a set $S$ that yields an acceptable value for the goodness of the workload.

The first approach termed *pre-paid* filling, determines a fixed set of $S$ distinct user-query pairs. This set is determined using a greedy approach that avails of the initial user- and query-similarities provided by the similarity model. Since this set is computed only once, this approach is non-incremental in behavior and lends itself to be an efficient solution to the Workload Filling problem.

The second approach, called *pay-as-you-go* filling, is an incremental solution to the Workload Filling problem. This approach undergoes a series of iterations, and in each iteration, generates a set of $S'$ user-query pairs. Whenever a pair i.e., a user asking a query, belongs to this set ($S'$), the system obtains a ranking function for this pair and re-calculates the initial user- and query-similarities (provided by the similarity model) to take into account the changes in the similarities that might result due to the addition of this deduced function to the matrix. Based on these new values, it determines another set $S'$ and continues this iterative process until the desired set $S$ is obtained.

Determining which of these two approaches to employ depends on the overall frequency of queries asked by users on the Web database as well as the nature of the queries, users and the deduced ranking functions (i.e., whether query and user similarity values change frequently based on addition of ranking functions to the workload). Figure 1 tries to distinguish the applicability of these approaches in different settings.
For instance, in a Web database scenario, if the frequency of queries received from users is low and, the corresponding similarities between users and queries tend to change drastically each time a ranking function is derived, the pay-as-you-go approach would be useful. In contrast, if a high frequency of queries are received without significant and frequent changes in the resulting user- and query-similarities, the pre-paid approach would prove to be better. Thus, the utility of the pre-paid approach diminishes as the frequency of user queries decreases with constant changes in user and query similarities, and vice versa for the pay-as-you-go approach.

Contributions: The contributions of this paper are -

- The proposed work addresses the question of establishing an appropriate workload used in user- and query-dependent ranking frameworks for Web databases.
- Given the NP-complete nature of the problem, we propose an approximate solution for determining the set of $S$ user-query pairs, from which ranking functions can be obtained, to fill the workload. We substantiate the nature of this filling by proposing a novel measure of workload goodness.
- Since the optimality of the approximate solution is intractable, we propose two approaches derived from greedy algorithms that respectively compute the necessary set of user-query pairs.
- We report the experimental results that establishes the quality and efficiency of our proposal for establishing appropriate workloads.

Roadmap: In Section 2, we overview the similarity model for user- and query-dependent ranking (proposed in [28]). Section 3 formally introduces the Workload Filling problem. It discusses the NP-complete nature of the solution to this problem, and introduces the metric of Workload Goodness that determines the quality of the approximate solution. In Section 4, we elaborate on the two greedy approaches that compute the necessary set of user-query pairs. The results of our experimental evaluation in terms of quality and efficiency are discussed in Section 5. Section 6 surveys the related work and Section 7 concludes the paper.

2 Preliminaries

In this section, we briefly review the similarity Model proposed for user- and query-dependent ranking in Web databases. This model (shown in Figure 2) [28] relies on the measures of query and user similarity. In addition, it employs a workload ($W$) comprising of ranking functions derived for several user-query pairs.

At the time of answering a query $Q_j$ asked by a user $U_i$, the framework employs the query-similarity model to establish an ordering of the queries based on their similarity to $Q_j$. Likewise, the users in the workload are ordered based on their respective similarities, estimated by the user-similarity model, with $U_i$. Based on this ordering, each user-query pair in the workload is assigned a rank based on its combined similarity with the pair $(U_i, Q_j)$. For instance, if an user $U_1$ and a query $Q_1$ occur as the first elements in the
<table>
<thead>
<tr>
<th>User-Query Pairs</th>
<th>Rank (w.r.t. $U_i, Q_j$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$U_i, Q_j$</td>
<td>$0+0=0$</td>
</tr>
<tr>
<td>$U_i, Q_1$</td>
<td>$0+1=1$</td>
</tr>
<tr>
<td>$U_1, Q_j$</td>
<td>$1+0=1$</td>
</tr>
<tr>
<td>$U_1, Q_1$</td>
<td>$1+1=2$</td>
</tr>
<tr>
<td>$U_i, Q_y$</td>
<td>$0+y=y$</td>
</tr>
<tr>
<td>$U_x, Q_j$</td>
<td>$x+0=x$</td>
</tr>
<tr>
<td>$U_x, Q_y$</td>
<td>$x+y$</td>
</tr>
</tbody>
</table>

Table 2: Sample Ranked List of User-Query Pairs

respective ordering of users and queries based on their similarities with the input pair, the pair $(U_1, Q_1)$ are assigned a rank that is an aggregate of their occurrence in the ordered lists (in this case, a rank of 2 will be assigned). For the workload shown in Table 1, if a sample ordering of queries based on their similarity with $Q_j$ is \{Q_j, Q_1, ..., Q_y\}, and the ordering of users based on their similarity with $U_i$ is \{U_i, U_1, ..., U_x\}, then the corresponding ranked list of user-query pairs with respect to the pair $(U_i, Q_j)$ is shown in Table 2.

Based on the ranks associated with each user-query pair with respect to the input pair, the most similar pair (e.g., $U_x, Q_y$) for which a ranking function ($\mathcal{F}_{xy}$) exists in the workload is selected to rank the results of $Q_j$ for $U_i$. The similarity model has been shown to achieve a good quality of ranking for the input user-query pair $U_i$ & $Q_j$ if the pair $(U_x, Q_y)$, from which the ranking function is selected, occurs in the set of top-$K$ pairs most similar to the input pair. Below, we briefly elaborate the details of these individual similarity models.

**Query similarity Model:** This model is based on the hypothesis — “if an input query $Q_j$ is most similar to a query $Q_y$ (in $U_i$’s workload), $U_i$ would display similar ranking preferences over the results of both queries; thus, the ranking function ($\mathcal{F}_{iy}$) derived for $Q_y$ can be used to rank $Q_j$’s results”. The similarity between two queries (in this case $Q_j$ and $Q_y$) is determined by comparing the attribute values in the query conditions.

**Definition** Given two queries $Q$ and $Q'$, each with the conjunctive selection conditions, respectively, of the form “WHERE $A_1=a_1$ AND · · · AND $A_m=a_m$” and “WHERE $A_1=a_1'$ AND · · · AND $A_m=a_m'$”, the query similarity between $Q$ and $Q'$ is given as theconjunctive similarities between the values $a_i$ and $a_i'$ for every attribute $A_i$ (Equation 1).

$$similarity(Q, Q') = \prod_{i=1}^{m} \text{sim}(Q[A_i = a_i], Q'[A_i = a_i'])$$ (1)

The similarity between any two values $a_1$ and $a_2$ for an attribute $A_i$ is determined as follows. Two queries $Q_{a_1}$ and $Q_{a_2}$ with the respective selection conditions: “WHERE $A_i = a_1$” and “WHERE $A_i = a_2$” are generated. Let $N_{a_1}$ and $N_{a_2}$ be the respective set of results obtained from the database for these two queries. The similarity between $a_1$ and $a_2$ is then given as the similarity between $N_{a_1}$ and $N_{a_2}$ (determined using the variant of the cosine-similarity [2]). Given two tuples $T = < t_1, t_2, ..., t_m >$ in $N_{a_1}$ and $T' = < t_1', t_2', ..., t_m' >$ in $N_{a_2}$, the similarity between $T$ and $T'$ is:

$$sim(T, T') = \sum_{i=1}^{m} \text{sim}(t_i, t_i')$$ (2)

where

$$\text{sim}(t_i, t_i') = \begin{cases} 1 & \text{if } t_i = t_i' \\ 0 & \text{if } t_i \neq t_i' \end{cases}$$ (3)

Using Equation 2, the similarity between the two sets $N_{a_1}$ and $N_{a_2}$ (which in turn, corresponds to the similarity between the values $a_1$ and $a_2$) is estimated as the average pair-wise similarity between the
tuples in $N_{a1}$ and $N_{a2}$ (Equation 4). These similarities between attribute values can then be substituted into Equation 1 to estimate query similarity.

$$sim(N_{a1}, N_{a2}) = \frac{\sum_{i=1}^{\left|N_{a1}\right|} \sum_{j=1}^{\left|N_{a2}\right|} sim(T_i, T_j)}{|N_{a1}| \cdot |N_{a2}|}$$

(4)

**User similarity Model**: This model is based on the hypothesis – “if user $U_i$ is similar to an existing user $U_x$, then, for the results of a given query (say $Q_j$), both users will show similar ranking preferences; therefore, $U_x$’s ranking function ($F_{xj}$) can be used to rank $Q_j$’s results for $U_i$ as well”. This hypothesis is translated to a formal model that determines similarity between a pair of users based on the similarity of their individual ranking functions over different common queries in the workload.

**Definition** Given two users $U_1$ and $U_2$ with the set of common queries – $\{Q_1, Q_2, ..., Q_r\}$, for which ranking functions ($\{F_{11}, F_{12}, ..., F_{1r}\}$ and $\{F_{21}, F_{22}, ..., F_{2r}\}$) exist in the workload, the user similarity between $U_1$ and $U_2$ is expressed as the average similarity between their individual ranking functions for each query $Q_p$ (shown in Equation 5):

$$similarity(U_1, U_2) = \frac{\sum_r sim(F_{1p}, F_{2p})}{r}$$

(5)

The similarity between a given pair of ranking functions is determined using the *Spearman’s rank correlation coefficient* ($\rho$) as follows. Consider two functions $F_{1p}$ and $F_{2p}$ derived for a pair of users for the same query $Q_p$ that has results $N_p$. The two functions are applied individually on $N_p$ to obtain two ranked sets of results – $N_{R1p}$ and $N_{R2p}$. If the number of tuples in the result sets is $N_p$, and $d_i$ is the difference between the ranks of the same tuple ($t_i$) in $N_{R1p}$ and $N_{R2p}$, then the similarity between $F_{1p}$ and $F_{2p}$ is given as the *Spearman’s rank correlation coefficient* given by Equation 6:

$$sim(F_{1p}, F_{2p}) = 1 - \frac{6 \cdot \sum_{i=1}^{N_p} d_i^2}{N_p^2 \cdot (N_p^2 - 1)}$$

(6)

### 3 Problem Definition

Consider a Web database table $D$, and let $W$ be a $M \times N$ workload matrix (e.g., Table 1) representing a large number of queries ($Q_1, Q_2, ..., Q_M$) and users ($U_1, U_2, ..., U_N$) across which pairwise query- and user-similarities have been established apriori. Each cell (e.g., [x,y]) in this matrix represents a ranking function ($F_{xy}$) for a specific user-query pair (such as $U_x$ and $Q_y$). Based on this setting, the **Workload Filling problem** is defined formally as:
**Definition** Given the workload matrix $W$ and a number $|S|$ (indicating the total number of ranking functions that can be collected), determine the set $S$ of user-query pairs such that –

1. for each user-query pair in $W$, there exists at least one user-query pair in the set $S$ that occurs in the list of the former’s top-$K$ most similar pairs.

2. a ranking function can be obtained from each user-query pair in this set $S$, and

**Assumptions:** It can be argued, based on the similarity model’s definition of user-similarity, that establishing user-similarities between every pair of users (across all $N$ users) may not be possible due to the fact that not all users may have asked the same query/queries. For the sake of consistency, in this work, we assume that pairwise user-similarities have been computed across all users. One mechanism to achieve this would be that the system (at the time of starting) may motivate the users, by providing some incentive as done by databases such as Amazon or Google, to provide a function for a specific query from which the similarities are computed. An alternative would be to establish an initial set of user similarities by comparing user profiles. Although user profiles are not considered by the current similarity model, we are currently working on extending this model to incorporate the knowledge from such profiles. Thus, for the rest of this work, we assume the existence of pairwise similarities across all $N$ users.

**Theorem 3.1** The Workload Filling problem is NP-Complete.

**Proof** In order to prove NP-Completeness, we shall reduce from the Dominating Set problem [15] [3]: “Given a graph $G = (V, E)$ consisting of $V$ vertices and $E$ edges, determine the smallest subset $D \subseteq V$ such that for every vertex not in $D$, there exists an edge connecting it to at least one vertex in $D$.”

Determining the minimum set $D$ is shown to be a NP-Hard problem by reducing it to the Set-covering problem [29]. Further, the decision version of this problem i.e., determining if there exists a subset $D$ of size $S$ has been proved to be a NP-Complete problem [3].

We cast the Workload Filling problem as follows: Consider the workload matrix $W$ comprising of a total of $M \times N$ cells ($\{c_1, c_2, ..., c_{M+N}\}$). Let this matrix be represented by a graph $G = (V, E)$, where each vertex $V_s$ represents a cell $c_s (\in W)$. For each vertex (e.g., $V_s$), there exist a total of $K$ edges connecting $V_s$ to its top-$K$ (including itself) most similar vertices. The goal is to determine the set $S$ of vertices such for each vertex in the graph, there is an edge connecting it to at least one vertex in the set $S$. This task is an equivalent to the task of determining a subset of size $|S|$ in the decision version of the Dominating Set problem, and hence is NP-complete.

Hence, we propose an approximate solution that finds set $S$ of user-query pairs to assist the similarity model. In order to validate the quality of the set produced by this solution, we propose a metric called Workload Goodness (or $(G(W,S))$). The motivation behind this metric is to determine, given $W$ and a set of ranking functions collected for $S$ distinct user-query pairs, the overall goodness of these functions in assisting the similarity model to rank the results of user-query pairs for which no functions exist. Below, we introduce the intuition for this metric, and then provide the formal definition for determining its value.

Consider the user-query pair $(U_i, Q_j)$ from Table 1 for which no ranking function exists, and let $W$ currently contain ranking functions for $S$ distinct user-query pairs. Based on the rank associated with each pair (with respect to the input pair) in this set $S$, the similarity model will pick the ranking function corresponding to the pair with the lowest rank (say $U_x, Q_y$ having a rank of ‘x+y’) $^3$.

Lower the rank of the selected pair, greater is the similarity of this pair with the input pair; thus, ensuring a good quality of the final ranking. In contrast, if the selected pair has a very large rank, its similarity with the input pair is less; thus, leading to an inferior quality of ranking produced by the corresponding ranking

$^3$As shown in Table 2, for a matrix of size $M \times N$, the ranks of the pairs in the set $S$ w.r.t., the input pair $(U_i, Q_j)$ can vary from a lowest value of 1 to a highest value of $M+N$
Definition Given the workload matrix $W$ of $M$ users and $N$ queries, and ranking functions are available for the set $S$ of distinct user-query pairs, the overall Workload Goodness (in terms of the quality of ranking achieved by the similarity model) is given by Equation 7.

$$G(W,S) = \frac{\sum_{p=1}^{M+N-S} rank[c_r]}{M \times N}$$

where, the value associated with $rank[c_r]$ is the rank of the cell $c_r \in S$ most similar to the cell $c_p$. Further, for all the cells belonging to $S$, a ranking function exists; hence, for all these cells, the rank and thus, the goodness equals 0. Therefore, we do not include these $|S|$ cells in the summation functions for determining the goodness.

Clearly, if ranking functions exist for each and every user-query pair in the workload (i.e., a best-case scenario), the goodness for such a workload will be maximized. Based on Equation 7, such a scenario will yield a value of “0” for $G(W,S)$. In contrast, if there exists only a single ranking function (a worst-case scenario) in the workload, and the cell corresponding to this function is the least similar cell to every other cell in the workload, the goodness of this workload will be minimized and Equation 7 will yield a corresponding value of “$(M-1) \times (N-1)$” for $G(W,S)$.

Based on the above discussion, it is evident that any approximate solution to the Workload Filling problem should determine a set $S$ of user-query pairs such that the value of $G(W,S)$ (from Equation 7) is as small as possible. We show that determining such an optimal selection of cells that provides the minimum goodness value is intractable.

**Theorem 3.2** Finding the optimal $S$ is Intractable.

**Proof** Given a workload matrix $W$ containing ranking functions for $S$ distinct user-query pairs, we can determine the corresponding goodness $G(W,S)$ for this configuration from Equation 7. Furthermore, given two different sets of the same size ($|S|$) – $S_1$ and $S_2$ of user-query pairs, we can determine which set would assist the similarity model in the best possible way by selecting the set producing the smaller (among the two) goodness values.

Now, in order to determine the optimal set of cells that yield the best (i.e., the minimum) value for goodness, we would have to generate a total of $\binom{M+N}{|S|}$ individual sets of user-query pairs followed by determining and comparing their respective goodness values. Given that $M$ and $N$ will be very large for workloads of most Web databases, this computation would lead to a combinatorial explosion (since it is exponential on $|S|$) and hence is intractable.

In order to address this challenge, we propose two distinct approaches based on greedy and top-K heuristics that each determine an approximate solution for the Workload Filling problem. We present the details of these two approaches in the following Section.

### 4 Filling The Workload

In this Section, we discuss the two proposed approaches that aim towards minimizing the value of $G(W,S)$ by finding an approximate yet appropriate set $S$ of user-query pairs for the workload matrix $W$.  


Table 3: Workload Example Scenario

<table>
<thead>
<tr>
<th>Cell</th>
<th>Rank 0</th>
<th>Rank 1</th>
<th>Rank 2</th>
<th>Rank 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>c₁</td>
<td>c₁</td>
<td>c₂</td>
<td>c₃</td>
<td>c₄</td>
</tr>
<tr>
<td>c₂</td>
<td>c₂</td>
<td>c₃</td>
<td>c₁</td>
<td>c₄</td>
</tr>
<tr>
<td>c₃</td>
<td>c₃</td>
<td>c₄</td>
<td>c₁</td>
<td>c₂</td>
</tr>
<tr>
<td>c₄</td>
<td>c₄</td>
<td>c₃</td>
<td>c₁</td>
<td>c₂</td>
</tr>
</tbody>
</table>

4.1 Pre-paid Filling

The essence of this approach is to determine at the start, a fixed set $S$ of user-query pairs in a greedy fashion. This approach, as explained in Section 1, is non-incremental and tends to be useful when the frequency of queries asked by users on the database is extremely high and the similarities between users and queries do not change drastically whenever a single (or a small set of) ranking function is added to the existing workload. We propose two variants for this approach, each of which employs a separate greedy algorithm to establish the set $S$ for obtaining an acceptable value of goodness for the workload. In order to better illustrate the two proposed algorithms, we use the following scenario:

**Example-2:** Consider a sample workload matrix comprising of two users ($U_1$ and $U_2$) and two queries ($Q_1$ and $Q_2$). Correspondingly, the workload comprises of a total of four cells ($c_1$, $c_2$, $c_3$, $c_4$), each representing a possible ranking function between a distinct user-query pair. For any given cell (e.g., $c_1$ corresponding to the pair $U_1$, $Q_1$), a ranking of all the cells in the workload can be obtained by combining the ranks based on the individual user and query similarities provided by the Similarity model. Table 4.1 represents, for each cell in this workload, the ranking of all cells based on the corresponding similarities. Let us consider that we need to determine the requisite set $S$ such that $|S| = 2$.

4.1.1 Independent Set Selection Algorithm

The intuition behind this algorithm is to determine a set of cells (corresponding to user-query pairs) from the workload matrix $W$ such that the average rank of each cell in this set, with respect to all cells in the matrix, is as high as possible. For instance, consider the cells $c_2$ and $c_3$ from Table 4.1 (in Example-2). The cell $c_2$, although possesses a higher rank (of 1) with respect to $c_1$, its rank with respect to cells $c_3$ and $c_4$ is low (since its rank is 3 with respect to both these cells); thus, giving $c_2$ a relatively lower average rank across all cells. In contrast, the cell $c_3$ has a comparatively higher average rank across all cells (ranks of 2, 1, 0 and 1 with respect to $c_1$, $c_2$, $c_3$ and $c_4$ respectively). Correspondingly, using Equation 7, if $c_2$ is selected to represent set $S$, the goodness value achieved will be 1.75, whereas $c_3$ will provide a value of 1 for the workload goodness. Thus, selecting $c_3$ instead of $c_2$ will yield a better value of goodness for the workload.

Formally, consider a cell $c_s \in W$, and let $\text{Rank}(W, c_s)$ represent the average rank of a cell $c_s$ with respect to all the cells in the workload given as:

$$\text{Rank}(W, c_s) = \frac{\sum_{r=1}^{M+N} \text{rank}[c_s^r]}{M+N}$$

where $\text{rank}[c_s^r]$ represents the rank of cell $c_s$ with respect to the cell $c_r$. For instance, $\text{rank}[c_1^2]$ from Table 4.1 is “2”.

The process (shown in Algorithm 1) starts with an empty set $S$, and determines the average rank for each cell in the workload (using Equation 8). It then sorts all the cells based on these ranks in ascending order and selects the first $|S|$ cells from this sorted list. Since this algorithm computes the average ranks for each cell, and independently picks the cells, this is the Independent Set Selection Algorithm.

For instance, in the case of Example-2 where we need to determine a set of size $|S| = 2$, this algorithm will pick the cells $c_3$ and $c_1$ (whose average ranks are 1 and 1.5 respectively, and together yield a goodness
For instance, consider that a cell greedily picks a cell from the available cells at that instance, without considering the cells already selected. A cell with an average value of 0.5 from Equation 7) will be selected instead of $c_2$ and $c_4$ (each having an average rank of 1.75) for the scenario in Example 2.

An important aspect of this algorithms is that it selects the required user-query pairs independently i.e., it greedily picks a cell from the available cells at that instance, without considering the cells already selected. For instance, consider that a cell $c_3$, that has the lowest average ranks amongst all cells in the workload, is selected in the first iteration. Now, the algorithm will pick the cell with the next lowest average (e.g., $c_1$). However, it is possible that the combination of the cells $c_3$ and $c_1$ may not assist in providing a goodness value better than the one provided by the combination of $c_3$ and a cell $c_4$ (which may not have a very low average; but may have a lower average with respect to those cells for which $c_2$ has a relatively higher average). Hence, a more suitable approach would be to consider the combined effect of a collection of cells in improving the overall goodness of the workload. In fact, this forms the intuition of our second variant to the Pre-paid filling approach.

### 4.1.2 Cumulative Selection Algorithm

This algorithm, termed as the Cumulative Selection algorithm, commences in the same fashion as its Independent counterpart i.e., the first cell (e.g., $c_s$) selected in the set $S$ will be the one that has the lowest value for $Rank(W, c_s)$ (from Equation 8) amongst all cells in the workload. However, a subsequent cell (e.g., $c_t$) will be selected only if $Rank(W, c_s \cup c_t) < Rank(W, c_s)$ i.e., if the collective effect of both these cells improves the overall rank, and hence the corresponding goodness of the workload. Thus, given that the set $S$ currently contains $K$ cells, the next cell (e.g., $c_{k+1}$) will be added to $S$ only if $G(W, K \cup c_{k+1}) < G(W, K)$. The process of obtaining the resulting set of $S$ user-query pairs is shown in Algorithm 2.

We explain the steps of the algorithm in the context of Example 2. The algorithm starts with an empty set $S'$. The outermost For-loop will be executed depending on the size (in this case, 2) of $|S|$. The innermost (second) For-loop will consider the cells currently not in $S$ in order to determine if any of these cells can cumulatively improve the goodness. Initially, since $S$ is empty, all the cells are considered. The innermost (third) For-loop determines the cumulative rank of each cell (marked as $\ast$ in the algorithm). For instance, the cell $c_1$, at the end of this For-loop will get a rank of $((0+2+2+2)/4=) 1.5$. Similarly, the cells $c_2$, $c_3$, and $c_4$ will obtain ranks of 1.75, 1 and 1.75 respectively. The sorting process will select cell $c_3$ from this list and add it to the set $S$.

In the next iteration of the second For-loop, we have at our disposal, only the cells $c_1$, $c_2$, and $c_4$. Now consider, cell $c_1$ from this list. For the innermost For-loop, we need to determine if the combination of $c_1$ and the already selected $c_3$ can assist in improving the goodness. From Table 4.1, it is evident that selecting $c_1$ will improve the goodness for cell $c_1$ (i.e., $l = 1$ in the For-loop). However, for cells $c_2$, $c_3$, and $c_4$, selecting $c_1$ does not give any advantage over the currently selected $c_3$. Hence, selecting cells $c_1$ and $c_3$ will provide a cumulative goodness of $((0+1+0+1)/4 =) 0.5$. However, this selection is better than if the cells $c_3$ and $c_4$ were considered (which yield a goodness of 0.75). Thus, the algorithm will iteratively determine the

---

**Algorithm 1 Independent Selection Algorithm**

```plaintext
\[ S = \emptyset, S' = \emptyset \]

for \( i = 1 \) to \( M \times N \) do

% Using Equation 8

\[
\text{Calculate } \text{Rank}(W, c_1) \\
\text{Add } c_1 \text{ and } \text{Rank}(W, c_1) \text{ to } S'
\]

end for

% Based on Rank of each cell

Sort $S'$ Descending

Select the first $|S|$ cells from sorted $S'$ and add to $S$
```

---

value of 0.5 from Equation 7) will be selected instead of $c_2$ and $c_4$ (each having an average rank of 1.75) for the scenario in Example 2.

An important aspect of this algorithms is that it selects the required user-query pairs independently i.e., it greedily picks a cell from the available cells at that instance, without considering the cells already selected. For instance, consider that a cell $c_3$, that has the lowest average ranks amongst all cells in the workload, is selected in the first iteration. Now, the algorithm will pick the cell with the next lowest average (e.g., $c_1$). However, it is possible that the combination of the cells $c_3$ and $c_1$ may not assist in providing a goodness value better than the one provided by the combination of $c_3$ and a cell $c_4$ (which may not have a very low average; but may have a lower average with respect to those cells for which $c_4$ has a relatively higher average). Hence, a more suitable approach would be to consider the combined effect of a collection of cells in improving the overall goodness of the workload. In fact, this forms the intuition of our second variant to the Pre-paid filling approach.

### 4.1.2 Cumulative Selection Algorithm

This algorithm, termed as the Cumulative Selection algorithm, commences in the same fashion as its Independent counterpart i.e., the first cell (e.g., $c_s$) selected in the set $S$ will be the one that has the lowest value for $Rank(W, c_s)$ (from Equation 8) amongst all cells in the workload. However, a subsequent cell (e.g., $c_t$) will be selected only if $Rank(W, c_s \cup c_t) < Rank(W, c_s)$ i.e., if the collective effect of both these cells improves the overall rank, and hence the corresponding goodness of the workload. Thus, given that the set $S$ currently contains $K$ cells, the next cell (e.g., $c_{k+1}$) will be added to $S$ only if $G(W, K \cup c_{k+1}) < G(W, K)$. The process of obtaining the resulting set of $S$ user-query pairs is shown in Algorithm 2.

We explain the steps of the algorithm in the context of Example 2. The algorithm starts with an empty set $S'$. The outermost For-loop will be executed depending on the size (in this case, 2) of $|S|$. The innermost (second) For-loop will consider the cells currently not in $S$ in order to determine if any of these cells can cumulatively improve the goodness. Initially, since $S$ is empty, all the cells are considered. The innermost (third) For-loop determines the cumulative rank of each cell (marked as $\ast$ in the algorithm). For instance, the cell $c_1$, at the end of this For-loop will get a rank of $((0+2+2+2)/4=) 1.5$. Similarly, the cells $c_2$, $c_3$, and $c_4$ will obtain ranks of 1.75, 1 and 1.75 respectively. The sorting process will select cell $c_3$ from this list and add it to the set $S$.

In the next iteration of the second For-loop, we have at our disposal, only the cells $c_1$, $c_2$, and $c_4$. Now consider, cell $c_1$ from this list. For the innermost For-loop, we need to determine if the combination of $c_1$ and the already selected $c_3$ can assist in improving the goodness. From Table 4.1, it is evident that selecting $c_1$ will improve the goodness for cell $c_1$ (i.e., $l = 1$ in the For-loop). However, for cells $c_2$, $c_3$, and $c_4$, selecting $c_1$ does not give any advantage over the currently selected $c_3$. Hence, selecting cells $c_1$ and $c_3$ will provide a cumulative goodness of $((0+1+0+1)/4 =) 0.5$. However, this selection is better than if the cells $c_3$ and $c_4$ were considered (which yield a goodness of 0.75). Thus, the algorithm will iteratively determine the
Algorithm 2 Cumulative Selection Algorithm

\[ S = \emptyset, \quad S' = \emptyset \]

for \( i = 1 \) to \(|S|\) do

\[ \text{for } j = 1 \text{ to } |M \ast N| - |S| \text{ do} \]

\[ \text{Rank}(W, c_j) = 0 \]

\[ \text{for } l = 1 \text{ to } |M \ast N| \text{ do} \]

\[ \text{Rank}(W, c_j) = \text{Rank}(W, c_j) + \min(\text{Rank}(c_j, c_l), \text{Rank}(S, c_l)) \]

end for

\[ \text{Rank}(W, c_j) = \text{Rank}(W, c_j)/M \ast N \]

Add \( c_j \) and \( \text{Rank}(W, c_j) \) to \( S' \)

end for

%%% Based on Rank of each cell %%

Sort \( S' \) Descending

Select first cell from sorted \( S' \) and add to \( S \)

end for

%%% Determining \( \text{Rank}(S, c_l) \) %%

\[ T = \emptyset \]

for \( p = 1 \) to \(|S|\) do

Determine \( \text{Rank}(c_p, c_l) \)

Add \( c_p \) and \( \text{Rank}(c_p, c_l) \) to \( T \)

end for

%%% Based on Rank of each cell %%

Sort \( T \) Descending

Select first cell (\( c_{T_1} \)) from sorted \( T \)

\[ \text{Rank}(S, c_l) = \text{Rank}(c_{T_1}, c_l) \]

final set \( S \) of user-query pairs that will cumulatively improve the goodness of the workload as desired.

In the case when no cells, from the set of cells currently available for selection in the innermost For-loop, assist in improving the goodness in a cumulative manner, the algorithm will be forced to pick the cells which have the highest average rank (from amongst the available cells) and add it to \( S \). In such cases, this algorithm will degenerate into the Independent Selection algorithm.

Given the presence of multiple For-loops, it is evident that the Cumulative Selection algorithm will tend to be more expensive than its Independent Selection counterpart. For instance, assuming that the user- and query- similarities along with the cell similarities and rankings can be computed apriori by using the Similarity model, the Independent selection algorithm typically involves the cost of determining the average ranks of each cell \( (O(M \ast N)^2) \) and the cost of ranking and obtaining the top \(|S|\) cells \( (O((M \ast N)\log(M \ast N)) \). The former cost can easily be reduced to incur a linear (instead of quadratic) time complexity by pushing the operation of finding the average ranks in the pre-computation step i.e., while finding the cell similarities and ranks, the average rank as well as the presence/absence of the cell in the top \(|S|\) similar cells can be calculated. Similarly, the cost of determining the set \( S \) of user-query pairs can be minimized to \( (O(M \ast N)\log(|S|)) \) by storing the cells and their ranks in an efficient structure such as a heap or a self-balancing binary search tree of size \(|S|\).

The Cumulative selection algorithm, in contrast, is computationally expensive due to the large number of cells involved in the iterations. Although determining the initial average rank of each cell can be pre-computed, due to the cumulative nature of the process, comparing ranks between cells in \( S \) and the ones in \( W \) need to be performed.
Algorithm 3 Pay-as-you-go Algorithm (High-level view)

\[ S = \emptyset, S' = \emptyset \]

for \( i = 1 \) to \(|S|\) do

STEP ONE:

\%
Determine pairs for \( S' \)

Determine the set \( S' \) using Algorithm 1

if No pair in \( S' \) has an existing function in \( W \) then

Obtain \( F_{ij} \) for at least one pair (e.g., \( U_i, Q_j \))

Proceed to STEP TWO

end if

STEP TWO:

\%
Re-arranging cell ordering

Recompute User Similarity between \( U_i \) and existing users

Adjust ranks of cells for which user similarities change

end for

It can be argued that since both algorithms determine a single and fixed set \( S \) of user-query pairs, the entire cost for this operation will be incurred only once. Furthermore, as our experimental results show, the additional cost incurred for the cumulative algorithm provides a significant improvement in the goodness over the independent algorithm.

4.2 The Pay-as-you-go Filling Approach

Our second proposed approach determines the set \( S \) of user-query pairs in an incremental fashion i.e., unlike the Pre-paid approach that determines the set \( S \) at the onset, this approach establishes the required set of pairs based on a number of a iterations (typically \(|S|\)). This approach is aimed towards those database applications where the frequency of queries asked by user is low; however, addition of a single (or a set of) ranking function to the existing workload leads to significant changes in the user similarities (and thus, changes in the rankings of user-query pairs with the remaining pairs in the workload).

Correspondingly, in each iteration, this approach determines a distinct set \( S' \) of user-query pairs. When a pair i.e., a user asking a query belongs to this set \( S' \), the necessary ranking function is obtained for this pair. This ranking function, when added to the existing workload, may lead to changes in user similarities. Consequently, changes in user similarities may lead to a change in the order of ranked cells for a number of cells in the matrix. Thus, the high-level algorithm for this approach (shown in Algorithm 3) can be split into two steps: i) Determine the set \( S' \) of user-query pairs in each iteration, and ii) Re-calculate the user similarities and adjust the ranked list of cells to reflect these changes before the next iteration.

As the algorithm shows, the Independent Selection algorithm (Section 4.1.1) can be employed for the process of determining the set \( S' \) of pairs in each iteration i.e., selecting the top-\(|S'|\) cells from the workload matrix that have the lowest average rank across all cells in the workload. The primary challenge of this approach is to ensure that the second step (i.e., re-arranging ordering of cells based on changes in similarity) is handled in an efficient manner. Specifically, we need to tackle two question – i) given that an additional ranking function is added to the workload, how many changes in the ordering of users based on the change in similarities with respect to a given user will occur?, and ii) how to ensure that these changes and the resulting re-computations are performed efficiently?

Consider Figure 3 that shows a simplified workload derived from the workload in Table 1. Let us consider the pair \((U_x, Q_a)\) corresponding to the cell \(c_1\), and let us assume that \(U_y\) is more similar to \(U_x\) than

\[ 4\text{As elaborated in Section 2, only the user similarity depends on the ranking functions in the workload. The query similarity computations are independent of the workload, and hence do not change.} \]
$U_i$. Similarly, $Q_b$ is more similar to $Q_a$ than $Q_j$. Correspondingly, all cells in the workload can be ranked with respect to cell $c_1$ as shown in the figure. Let us presume that the set $S'$, generated by Algorithm 3 in the first iteration, contains the cell $c_9$ (corresponding to the pair $U_i, Q_j$ for which no ranking function exists originally) for which we subsequently obtain a ranking function. Based on this function, if the similarity between $U_x$ and $U_i$ changes such that $U_i$ becomes more similar to $U_x$ than $U_y$ (as shown in the Modified workload in the figure), then the ordering of users with respect to $U_x$ changes leading to a change in the ordering of certain cells with respect to $c_1$ also undergoes a change. As seen in the figure, with respect to $U_x$ the order of both users has changed. Thus with respect to each cell associated with the user $U_x$ (i.e., $c_2, c_3$ and $c_3$), we need to recalculate the ranks of all cells in the workload in order to ensure a correct cell ordering for the next iteration. The changed order of cells for the cell $c_1$ is shown in Figure 3.

Thus, before the first iteration, let us assume that the workload matrix $W$ contains a total of $|F|$ ranking functions. On addition of an additional ranking function to the workload, the maximum changes in ordering of users (assuming a worst case scenario when all $|F|$ users have asked the same query) respect to their similarity to a given user $U_x$ is $|F|^2$. Consequently, we need to recalculate the order and rank of each cell in the workload with respect to a total of $|F|^2 \times M$ cells.

In order to carry this process efficiently, we maintain, for each user (say $U_x$), a priority queue containing all the users sorted based on their ranks with $U_x$. Whenever, the similarity between a user (say $U_i$) and $U_x$ changes, the changes in the respective ranks can be performed in $O(\log N)$ time. Correspondingly, for each cell (which is a combination of a user and query), we maintain a separate priority queue, and based on changes in the ranks of users, the ranks for each cell can be obtained in $O(M \times \log N)$ time.

In the context of Web databases, it is unlikely that user similarities will change drastically every time a user asks a query. Consequently, the Pay-as-you-go algorithm can be adjusted to calculate user similarities after certain cells in the set $S'$ are filled; thus, the process of finding a fresh set of candidate cells and re-arranging cell ranks can be deferred to specific intervals of time instead of performing it in each iteration.

## 5 Experimental Evaluation

We have evaluated the algorithms for each proposed approach (Pre-paid and Pay-as-you-go) for quality/accuracy in terms of their respective estimated value of workload goodness $G(W, S)$. Additionally, we also evaluated the efficiency of each of these algorithms in terms of the time they each take to yield the requisite set of $S$ user-query pairs.
5.1 Setup

The workload matrix \((W)\) is made up of users asking queries on a Web database. Currently, no Web database provide such a workload that can be used for experimental evaluation. To obtain a real workload using real Web users and queries, we would have to convert our framework into a full-fledged Web database application and invite a large number of users to pose queries to the system – a task beyond the scope of the paper. Hence, in order to experimentally validate the quality of our proposal, we had to rely on generating a synthetic workload of users and queries. For this, we relied on the two databases provided by Google Base [14]. The first is a vehicle database comprising of 8 distinct attributes (Make, Model, Vehicle-Type, Mileage, Price, Color, Location, and Transmission). The second is a real estate database with 12 attributes such as Location, Price, House Area, Bedrooms, Bathrooms, etc.. Google provides APIs for querying its databases, and returns a maximum of 5000 results for every query asked via the API to either database.

On each database, we generated 1 million queries using an assumption that queries asked by users on Web databases follow the Zipf’s Law distribution [23] i.e., queries comprising of a lesser number of attributes over a larger domain of values are asked more frequently than queries with larger number of attributes. Correspondingly, these queries form one dimension of the matrix \(W\). We then employed the query similarity component of the Similarity model (see Figure 2) to determine the individual pairwise similarities between these queries. Since obtaining data of real users is difficult, we generated a total of 1 million synthetic users, thus, forming the user dimension of \(W\). In order to determine the requisite user similarity between each pair of users, we randomly filled a small percentage (1%, 10%, 20%, etc.) of this workload with ranking functions. These functions were designed (by varying the variables) from an existing collection of ranking functions obtained via a survey conducted on real users over these two Web databases in [27] [28].

Our experiments were performed on a 2.6 GHz AMD Opteron Quad Processor machine with 16GB RAM running on a 64-bit Redhat Linux installation. All algorithms were implemented in Java.

5.2 Quality Evaluation

Given a \(M \times N\) workload matrix \(W\) comprising of \(M\) queries and \(N\) users, and the total number of ranking functions \(|S|\) that can be collected from distinct user-query pairs, we tested the quality of our proposed algorithms for either approaches, in terms of the value produced for the goodness of the workload (i.e., \(G(W, S)\)).

**Pre-paid Filling Approach:** As proved in Theorem 3.2, given a workload matrix of 1 million users and queries, determining the optimal set of \(S\) user-query pairs that provide the best value for workload goodness is intractable. Hence, we to validate our proposed algorithms, we randomly selected a set of 10 users and 10 queries (from the set of 1 million users and queries), to form a \(10 \times 10\) workload matrix of 100 cells. We then filled 1% of this matrix with ranking functions in order to obtain the necessary user similarities.

We set a value of \(|S| = 5\), and determined the optimal set \(S_{OPT}\) of user-query pairs that provide the best value of the workload goodness. We then applied each algorithm (Independent Selection and Cumulative Selection) over this workload, and determined the respective sets of \(S_{IND}\) and \(S_{CUM}\) user-query pairs. We then applied a random selection process and obtained a set \(S_{RAND}\) of pairs. For all these sets, we determined the corresponding goodness (\(G(10 \times 10, |5|)\)). We performed the same experiment by setting a value of \(S = 10\) to obtain the necessary goodness values across each algorithm. Figure 4 shows the corresponding results for this set of experiments over both databases.

It is obvious that the optimal set will produce the best value of goodness. Furthermore, our proposed algorithms perform yield a goodness that is closer to the optimal selection than a random selection of pairs from the matrix. Additionally, as intuitively motivated in Section 4.1.2, the Cumulative algorithm perform better than its Independent counterpart.
We then tested the quality of our algorithms for the entire workload matrix of 1 million users and queries. We performed three sets of experiments wherein we set the values of $|S|$ as 1000, 2000 and 5000 respectively. Since obtaining the optimal solution for such large numbers is intractable, we compared our approach to the K-Means clustering algorithm[18]. Given that for each user-query pair we can rank every user-query pair in the matrix, we clustered all the cells of this matrix, as done in [27].

For the experiment with $|S| = 1000$, we generated 1000 clusters and selected the centroid of each cluster as the user-query pair to be selected to be in the set $S_{K_Means}$. Similarly, 2000 and 5000 clusters were generated for the remaining two sets of experiments. Figure 5 shows the results, in terms of the goodness values obtained, for these sets of experiments across both databases. Again, as predicted, the Cumulative algorithm performs better the Independent counterpart. Furthermore, both these algorithms outperform the Clustering approach.

**Pay-as-you-go Filling Approach:** Similar to the previous approach, in order to test our algorithm against an optimal selection, we selected the same set of 10 queries and 10 users (as selected above) to form a $10 \times 10$ workload matrix. Similar to the previous approach, we filled 1% of this workload with ranking functions to determine the necessary user similarities, and performed two experiments by setting the size of $S$ i.e., the number of functions to obtain as 5 and 10 respectively. Furthermore, in each iteration of the workload,
we generated a set of $S' = 5$ user-query pairs (STEP ONE of Algorithm 3), and selected a random pair from this set to improve the goodness. In addition, we also determine the optimal pair in each iteration that improved the goodness in the maximum possible way.

Figure 6 shows the results of this set of experiments for both databases. Again, the optimal selection provides an ideal goodness value. However, the goodness determined by our proposed incremental algorithm is closer to this optimal value and considerably outperforms a random selection of pairs for this matrix.

We tested the quality of this algorithm over the entire workload of 1 million users and queries. We started with an initial set of 1000 ranking functions i.e., 1000 filled cells, and incrementally determined the set $S$ of user-query pairs (by setting $|S|$ as 100, 500, 1000 and 2000). We then performed an identical set of experiments by filling the workload with 2000 functions and incrementing $|S|$ in the same way. As Figure 7 shows, for both databases, the proposed algorithm performs significantly better than random selection process.

**Pre-paid v.s. Pay-as-you-go Filling Approach:** In this set of experiments, we compared the algorithms across both proposed approaches for quality i.e., the value of goodness estimated by these algorithms. We considered the entire workload of 1 Million users and queries and considered a 10% filling of the workload with ranking functions respectively (for estimating the necessary user similarities). We then pitted all the proposed algorithms against each other in determining the necessary goodness for the set $S$ by varying the size of this set from 100, 200, 250 and 500.

For the Pay-as-you-go approach, we generated, in each iteration, the set $S'$ of size 100. From this set, we selected the first, last and middle pair in each iteration to finally generate three distinct sets – $S_{\text{top1}}, S_{\text{bottom1}},$
and $S_{middle1}$. We estimated the goodness provided by each of these sets and compared it to the goodness determined by both algorithms in the Pre-paid approach. Figure 8 shows the results of this experiment across both databases. As the results show, the Cumulative Pre-paid approach is better for smaller sizes of set $S$ but as the size of this set increases, the Pay-as-you-go approach of selection the top-1 function in each iteration begins improving the goodness in the best possible way.

### 5.3 Efficiency Evaluation

The goal of this study was to determine whether the approaches proposed in this paper can be used for determining the requisite set of $S$ user-query pairs in a real-time application. The Pre-paid approach determines the set of user-query pairs once, and hence, involves only a one-time cost of deducing these pairs. However, the need for providing greedy solutions to this approach stems from the difficulty in obtaining the optimal solution for large workload matrices. For instance, it takes well over 20 hours (using an efficient multi-threaded program) to determine the optimal set of sizes $|S| = 5$ and $|S| = 10$ pairs respectively for a small matrix of $10 \times 10$. In contrast, the Independent approach takes less than 5 minutes whereas the Cumulative algorithm needs less than 30 minutes to determine the set of same sizes.

The Pay-as-you-go approach involves a heavy cost due to the re-computations of similarities performed in each iteration. However, by hashing the user- and query- similarity ranks, and managing the corresponding cell rankings using an efficient priority queue implementation brings down the time time involved in this process. Figure 9 shows, for both databases, the time (in seconds) involved in determining the set of sizes $|S| = 1000$ and $|S| = 2000$ user-query pairs for a $1M \times 1M$ workload matrix. As seen in the results, over a considerably large workload, the time to determine the cell rankings and user similarities dominate over the process of determining the set of $S$ pairs. As argued in Section 4.2, by deferring the computation of user similarities, instead of determining them whenever a user asks a query, the time to determine the requisite pairs can be substantially reduced; albeit by incurring a small loss of the quality of workload goodness.

### 6 Related Work

The task of establishing a workload, proposed in this paper, is in the context of user- and query-dependent ranking on Web databases. Although ranking has received significant attention in the domain of databases,
existing techniques either support only query-dependent ranking [2] [10] [11] [26] or only user-dependent ranking [19] [21] [17].

Some of the work in query-dependent ranking [10] [11] employs a simple workload of past queries in order to deduce a ranking function for the database. However, since a single ranking function is employed across all users, these frameworks do not need to pick particular queries that might assist in improving the quality of ranking. In fact, generating a considerably large corpus of queries proves to be sufficient in deducing an appropriate ranking function across all users for these techniques. Similarly, the workload employed by user-dependent ranking techniques [19] [21] is to collect sufficient information from the users (in terms of profiles) to provide customized ranking for each user. However, since these works do not assume that the same user may have different preferences towards the results of different queries, collecting a simple profile suffices these techniques. In contrast, the workload established in this paper is a collection of individual ranking functions across different user-query pairs that will assist in improving the quality of ranking for future queries asked by users. Hence, collecting a single profile or a large collection of random queries will not be much helpful to the problem at hand.

Similar to the workload established in this work, recommendation systems in the form of content-filtering [4], [6], [13] as well as collaborative-filtering [16] [5] [8] [12] employ a similar user-item rating matrix. A significant difference that separates our work from these techniques is the information contained in the matrix. While each cell in the rating matrix of recommendation systems contains information in the form of a simple rating given by an user to an object (e.g., rating a particular movie), the information contained in each cell of our workload represents is more complex in terms of a preference to individual tuples within the results of a query. While it is significantly easier, in terms of time and effort, to obtain a rating for an item, obtaining a ranking function from a user incurs considerable time and effort (since the user needs to browse through the query results and then make individual choices towards the relevance/non-relevance of the tuples). Furthermore, since the goal of recommendation systems is to recommend items to user based on past data, the more ratings obtained by this matrix, the better it serves these systems. Although this is applicable even in the case of our workload for ranking in Web databases, owing to the high cost in obtaining individual functions, the number of functions that can be possibly obtained will be much lesser than the ratings obtained for a recommender system. Consequently, deciding the exact pairs for whom the functions are to obtained (for ensuring a good quality of subsequent ranking) is much more vital than determining the objects for which ratings can be obtained.

To the best of our knowledge, techniques for establishing workloads of ranking functions to assists ranking in databases has not been addressed in literature.
7 Conclusion

In this paper, we proposed a solution to the problem of establishing a workload of ranking functions for assisting user- and query-dependent ranking on Web databases. Given the NP-complete nature of the optimal solution to this problem, we proposed two approaches, Pre-paid and Pay-as-you-go based on greedy algorithms, for determining an acceptable set of user-query pairs whose respective ranking functions would then represent the workload. In order to validate the quality of the solution, we introduced a novel metric of Workload Goodness that determines the contribution of the estimated set of pairs to improving the overall quality of ranking by the Similarity model. We analytically explained the effectiveness of our proposal and validated it experimentally over two real Web databases.

Our work brings forth several additional challenges. Although this work assumes that for the user-query pairs selected by the respective algorithms, the corresponding ranking functions can be obtained, techniques for inferring these ranking functions based on user’s preferences need to be further investigated. Furthermore, determining the minimum number of functions (i.e., |S|) that can ensure an acceptable goodness for the workload requires further study. Applicability of this approach for establishing workloads in additional domains and applications needs to be explored.

References


