APMD-Workbench: A Benchmark for Query Personalization

Verónika Peralta
Laboratoire d'Informatique
Université de Tours
3, pl Jean Jaures
41000 Blois, France
+33-2-54552112
vperalta@univ-tours.fr

Dimitre Kostadinov
Alcatel-Lucent Bell Labs France
Route de Villejust
91620 Nozay, France
+33-1-30772116
Dimitre_Davidov.Kostadinov@alcatel-lucent.com

Mokrane Bouzeghoub
Laboratoire PRiSM
Université de Versailles
45, av des Etats-unis
78000 Versailles, France
+33-1-39254057
mok@prism.uvsq.fr

ABSTRACT

Query personalization algorithms intend to deliver the most relevant data to each user according to their profiles. Validating efficiency and relevancy of such algorithms still remains a very difficult task as it requires a scalable dataset, a bunch of user profiles and queries and possibly user feedbacks. At the best of our knowledge, there is not a reference benchmark devoted to the validation of such algorithms. In most of the published papers, validation of personalized queries is done through ad hoc benchmaks whose features are not given and which are generally not provided to other authors to perform similar evaluations. In this paper, we present a benchmark for query personalization which aims to be a reference to validate query personalization systems. The benchmark is based on a large test bed derived from MovieLens and IMDb datasets, and provides, besides the data set itself, a large sample of queries and users as well as their corresponding good results.

1. INTRODUCTION

Query personalization is one of the main solutions to improve data relevance in information retrieval and database systems. Before being executed, user queries are reformulated on the basis of user profile preferences. This allows targeting user's domain of interest and thus delivering pertinent results and reducing result size.

In order to measure the relevance of results, we need to compare delivered results with those effectively preferred by users. In other words, we need a reference data set that contains several queries and the corresponding sets of query results that are relevant for each user. In this way, we can quantitatively evaluate the behaviour of a personalization algorithm using metrics such as precision, recall, result size, performance, etc.

In this paper we describe the construction of a benchmark for query personalization. There exist several benchmarks among which we can cite the TPC benchmarks for database server performances [9] or the TREC benchmarks for information retrieval systems [8]. However, as far as we know, there is no benchmark providing a validation framework to query personalization algorithms, at least in the database domain. A benchmark for query personalization should also manage different users and their preferences. Specifically, they should provide a large database, a set of users, a set of queries and the reference results associated to each user and query, i.e. they should provide collections of triplets {(query, user) → results}.

Obtaining such reference results is very costly because it involves asking users to explicitly evaluate query results. The TREC benchmark [8] was built in this manner and the task lasted several years. In addition, there was no notion of user profile in TREC, but users divided the task of judging if documents were pertinent or not in a global manner. For the proposed benchmark, instead of asking a set of users to manually classify query results according to their preferences and feelings, we reuse ratings already expressed by real users and published on the Web.

Our dataset is derived from two public databases, i.e. MovieLens [1] and IMDb [5]. Both databases deal with data about movies. The IMDb database contains rich information about films, actors, directors, the places where they are produced, their budgets, their categories and the average rank given by the users who had evaluated them. The MovieLens database contains very few information about films but provides a huge amount of evaluations given by users who have seen these films. The two databases are complementary as they almost target the same movies (actually the set of films referred in MovieLens is a subset of those referred in IMDb). The main advantage of using these databases is their large volume of data, which is freely available through Internet. In addition, movie data is very easy to understand, to use and to analyze.

The construction of the benchmark proceeds in three phases, as illustrated in Figure 1. The first phase consists in extracting, transforming and loading movie data from IMDb and MovieLens. The second phase consists in generating a set of queries and deriving the corresponding "good" results for each user. This can be done

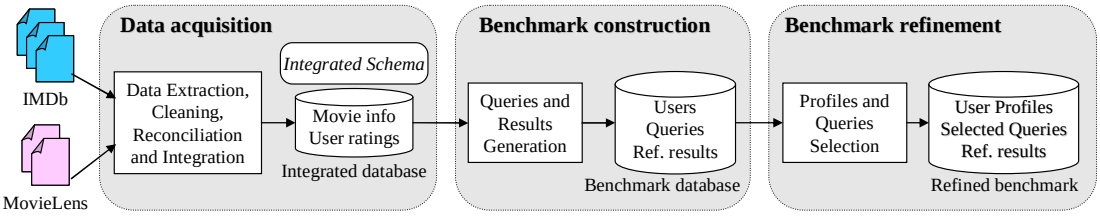

Figure 1 – Benchmark construction process

thanks to the content originated from MovieLens which provides a large set of user evaluations on different films they have seen. The benchmark database contains several thousand queries and users, and several millions reference results. The third phase consists in selecting the users and queries that are worth to be tested in a particular type of test. This selection is particularly necessary as the execution of the whole set of queries for the whole set of users necessitates tens of years although limiting the query evaluation process to a few seconds. Actually, the benchmark database serves as a basement to generate several specific benchmarks with different features, depending of the evaluation goal.

This paper describes the construction of the benchmark and its use for query reformulation. Section 2 describes the procedures for extracting and integrating IMDb and MovieLens data, for generating a set of queries, and for obtaining the reference results for each pair (query, user). Section 3 presents an example of use of the benchmark, illustrating the generation of user profiles and the comparison of results for a particular type of tests. Finally, Section 4 presents our conclusions.

2. BENCHMARK DESIGN

This section describes the challenges, the difficulties and the strategies used to define the benchmark.

2.1 Data Acquisition

The first phase consists in extracting, transforming and integrating IMDb and MovieLens data in order to build a relational database about movies, which includes movie descriptions and user ratings.

MovieLens data set consists of 3 text files, with tabular format, describing 1.000.209 anonymous ratings of 3.883 movies made by 6.040 MovieLens users. IMDb data set consists of 49 ad-hoc text files, called lists, which record different details about movies. At October 2006, list sizes varied from 25.000 to 5.000.000 tuples about more than 850.000 movies. We extracted data from 23 lists, representing the most relevant tabular features about movies.

For both data sets, data extraction consisted in several tasks including loading of text data into a relational database, transformation and normalization of data types, standardization of codes, filtering of inconsistent values and duplicate elimination. The matching of MovieLens and IMDb movie titles (which identify movies) was the most difficult task in the construction of the integrated database.

The integrated schema consists in 52 tables describing movies, companies and persons related to movies as well as the users that evaluated movies. It includes 1 table listing movies, 1 table containing user evaluations, 3 tables describing users, 20 tables describing movie features (e.g. genres), 23 tables relating movies to features and 4 auxiliary tables. We refer the interested user to [6] for further details on data acquisition.

2.2 Generation of Queries

In order to build the benchmark database we need to build a set of queries and the reference results for each pair (query, user). Instead of asking users to manually classify query results according to their relevance, we reuse movie ratings already given by MovieLens users. Specifically, each tuple of the *I_UserRatings* table of the integrated database corresponds to an evaluation of a movie, registered by a user, indicating a rating (in a 1-5 star scale).

As ratings qualify movies, we generate a set of queries returning movies, which are lately compared to the movies having a high user rating. Queries have a star-like form:

SELECT I_UserRatings.movieid
FROM I_UserRatings, <*additional tables*>
WHERE <*filtering conditions*> AND <*join conditions*>

Note that instead of selecting movies from the whole set of movies, we consider as space of solutions, the ones that the user has already evaluated (i.e. the I_UserRatings table). In this way, queries return only movies whose ratings are already known. We join the I_UserRatings table with some tables describing movie features and we add some filtering conditions on such features.

Filtering conditions are predicates of the form *feature operator value*, where *operator* $\in \{=,\leq,\geq\}$, and *value* ranges in the domain of a movie *feature*. We randomly select a small number of predicates (from 1 to 5) for each query, which avoids generating monster queries that returns no data. However, the randomness of the selection allows obtaining result sets of different sizes, ranging from almost empty sets when queries have several restrictive predicates to almost all data when queries have few non-restrictive predicates. The additional tables are those referenced in the predicates and those necessary to join them to the *I_UserRatings* table. Details on query generation can be found in [7].

2.3 Computation of Reference Results

As the rating of each movie is known, we can easily build the set of reference results, i.e. those movies rated with 3, 4 or 5 stars.

Actually, we partition the I_UserRatings table into two subsets: (i) *training set*, available for the generation of user profiles, and (ii) *test set*, available for executing queries and measuring personalization results. This partitioning allows personalization algorithms to learn user preferences and derive user profiles without biasing the test results. To avoid side effects generated by the arbitrary choice of these two subsets, the process has been repeated for several subsets, randomly generated from the original dataset.

In order to compute partitions, five random attributes were added to the I_UserRatings table (namely, C1, C2, C3, C4 and C5), each one randomly filled with an integer between 0 and 9. Therefore, the test set is described by a condition on the values of one of the Ci attributes ($1 \leq i \leq 5$). We also parameterize the rating above which a film is considered to be good (from 3 to 5).

Consequently, in order to execute a query for a given user according to a partitioning and rating strategy, the query is restricted with three conditions on the I_UserRatings table: (i) a *TestSetCondition* of the form $Ci<N$, $1 \leq i \leq 5$, $0 \leq N \leq 9$, (ii) a *RatingCondition* of the form *rating* $\geq V$, $3 \leq V \leq 5$ and (iii) a UserCondition of the form *userid=U*, $1 \leq U \leq 6040$. As an example of restricted queries consider:

SELECT I_UserRatings.movieid
FROM I_UserRatings, I_MovieCountries
WHERE I_UserRatings.movieid =I_MovieCountries.movieid
AND I_MovieCountries.country = 'France'
AND *Ci<N* AND userid = *U* AND rating $\geq$ *V*;

2.4 Experiment and statistics

The benchmark was developed and stored in an Oracle 9i database. Preprocessing and parsing procedures where implemented in Java, loading was performed with the SQL-Loader utility and the remaining procedures where implemented in PL-SQL (stored procedures). In this section we describe some results and statistics obtained from the execution of those procedures, specifically, we describe parameter setting and we analyze the generated queries and their result sizes.

2.4.1 Setting parameters

As previously argued, we aim at generating different partitioning and rating strategies in order to obtain unbiased experimental results. We tested different parameters discarding those that produced too few tuples in the test set (e.g. when setting 'rating = 5').

We kept the 20 strategies shown in Table 1 (they are packed by 5, for $1 \le i \le 5$). These strategies combine two test set sizes, with approximately 50% (Ci≥5) and 70% (Ci≥3) of tuples respectively, and two rating conditions (*rating* ≥ *3* and *rating* ≥ *4*). Table 2 shows the average number of good ratings in the test set per user and type of strategy.

Table 1 – Combination of partition and rating strategies

Strategy id	Test set cond.	Rating cond.
1-5	Ci ≥ 5	Rating ≥ 3
6-10	Ci ≥ 5	Rating ≥ 4
11-15	Ci ≥ 3	Rating ≥ 3
16-20	Ci ≥ 3	Rating ≥ 4

Table 2 –Average number of good ratings in the training set per user and type of strategy

Test cond	All ratings	Rating≥3	Rating≥4
Ci ≥ 3	116	96	66
Ci ≥ 5	83	69	48

2.4.2 Obtained queries

Having set parameters, we proceeded to execute the query generation procedures. We obtained an initial set of 622.061 predicates for all queries, from which we randomly selected from 1 to 5 predicates per query. We generated 6040 queries and we added a query with no predicates. After generating queries, we executed them over several test sets (for all users) and we measured the size of the obtained results. Figure 2 illustrates result sizes for one particular strategy (with test size=70% and rating≥3). We took two measures: the average of user's result sizes, and the maximum result size (for the user obtaining the most of results). Note that most queries returns less than 20 tuples in average but they return more results for some users. Further details can be found in [7].

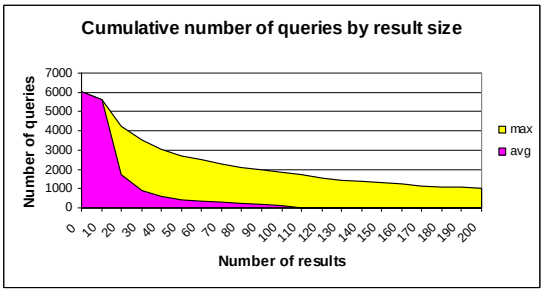

Figure 2 – Number of queries by result size

3. AN EXAMPLE OF BENCHMARK USE

The benchmark presented in the previous section can be used to evaluate various personalization approaches. This section shows its use for evaluating query reformulation approaches.

3.1 Query Reformulation Approaches

Query reformulation applies to the context of mediation systems. It consists in reinterpreting the user intention, expressed in his initial query, into a more complete query, considering at the same time the user profile and the descriptions of the data sources. Thus, query reformulation integrates two complementary tasks: (i) query enrichment which consists in integrating user profile content in the user query and (ii) query rewriting which transforms a query expressed on the virtual schema in expressions (rewritings) expressed on the data sources.

We evaluated three query reformulation approaches, proposed in [3]. Two of them are compositions of existing algorithms for query rewriting [2] and query enrichment [4]. They differ in the order of application of the rewriting and enrichment algorithms. The third approach is an interleaving of the two previous ones.

In order to evaluate and compare these query reformulation techniques, we adapted the benchmark presented in the previous section. Next sub-section presents the benchmark refinement process.

3.2 Benchmark Refinement

The query reformulation approaches require the presence of a distributed system and the availability of predicate-based user profiles. In order to fulfill these requirements, the benchmark have been refined by simulating a distributed system and by extracting profile predicates from the users' training sets.

To simulate a distributed environment, the integrated schema of the benchmark is taken as global schema. Then 52 views have been manually defined over this global schema in accordance with the following assumptions: (i) views should provide all data contained in the integrated database, assuring no information loss, (ii) views schemas should overlap, generating some redundancy in order to measure the capacity of an approach to select only relevant data sources, and (iii) some views definitions should contain selection predicates enabling to check if a reformulation approach selects data sources which are able to better satisfy the user preferences . Each view is considered as being a separate data source. User profiles are constructed as sets of predicates that state user preferences on movie features. Profile predicates have the form *feature=value*, where *value* ranges in the domain of the movie *feature*. For example, a certain user may prefer *movies spoken in French* or *action movies*; which is expressed by the predicates: *Language = French*, *Genre = Action*.

In order to extract a user profile from a set of user evaluations (those of the training set), we look for common features of the evaluated movies. For example, if most of the movies to which the user has assigned a great rating are filmed in France, we deduce that the user prefers *movies filmed in France*, and we propose the predicate *LocationCountry=France*.

We generated a large set of predicates where each predicate is associated with a *weight* representing the percentage of evaluated films that satisfy it. Weights allow conforming more or less restrictive user profiles by choosing the predicates with higher weights or accepting predicates with lower weights. Weighted predicates have the form *<table.attribute operator value (weight)>* where: *table* and *attribute* refer to an attribute of a table of the integrated schema (referencing a movie feature), *value* is an element of the attribute domain, *operator* ∈ {=,<,≤,>,≥} and *weight* represents the percentage of the evaluated films that satisfy the predicate. Examples of weighted predicates are "I_MovieLanguages.language = English (80)" and "I_Countries.continent = Europe (25)" which can be interpreted as *among the films the user has evaluated, 80% are spoken in English and 25% have been filmed in Europe*.

3.3 Query and Profile Selection

The execution of the whole set of available queries using the whole set of generated profiles necessitates tens of years although limiting the query reformulation process to a few seconds. Thus, the evaluation was performed on a subset of queries and profiles.

The main parameter which is taken into account for query selection is the response time of the query reformulation algorithms. Query rewriting is the most time-consuming phase of the reformulation process. According to [2], the response time of query rewriting algorithms depends on several parameters such as the number of virtual relations in the query, the schema type, the number of sources, the number of variables in the source schemas, etc. Most of these parameters do not vary in our benchmark; the only variable parameter is the number of virtual relations in the queries. We made some preliminary tests which shown that query rewriting takes about 1 minute for queries expressed on 9 virtual relations and more than 10 minutes for queries with 10 relations. As query reformulation is usually a real time process, we limited its response time to 1 minute. In addition, to enable the expansion of a query with additional virtual relations during query enrichment, we restricted to queries expressed on at most 5 virtual relations. Thus, a total of 13 queries was selected for the tests including the only available query expressed on 1 virtual relation and 3 queries for each other configuration (i.e. expressed on 2, 3, 4 and 5 virtual relations).
The selection of a subset of users (and their profiles) is guided by the following requirements: (i) a user test set should be large enough to get significant number of results when executing the queries on it, (ii) the predicates of a user profile should be expressed on different attributes (to simplify the query enrichment), (iii) profile predicates should have weights superior or equal to 80, and (iv) a user should have enough predicates to allow considering profiles with different cardinalities. To satisfy these requirements, we applied several filtering steps. First, users whose test set contain less than 100 movies have been pruned. Second, for each profile we pruned all predicates expressed on the same attribute but the one with the highest weight. Finally, only predicates which weights are superior or equal to 80 were selected. The filtering resulted in 747 profiles having from 2 to 10 predicates. For our tests we selected the 2 available profiles containing 10 predicates and 3 profiles containing 9 predicates. Each profile was then used to produce two new profiles by randomly selecting 3 and 6 of its predicates. Thus, the refined benchmark contains a total of 15 user profiles (3 profiles per user).

The benchmark allowed to compare the three query reformulation approaches and to highlight the contexts where each one performs better. Evaluations show that introducing personalization into query reformulation improves the precision of the obtained results but increases response time and can lead to the loss of relevant results. A more complete description of the tests and the obtained results can be found in [3].

4. CONCLUSIONS

This paper proposes a benchmark for query personalization based on movies rated by real users. The benchmark database includes a large set of users, queries and reference results.

The benchmark can be refined to support the evaluation of various personalization approaches. We showed an example of refinement that we used for comparing three query reformulation approaches. In this example we generated simple user profiles. The benchmark can also be used for testing and comparing profile generation algorithms, both for individual users and communities. Our hope is that this platform serves as a reference test in the database community in order to federate the sparse evaluations around a common data set as done for example in the TREC protocol.

A detailed description of the APMD-Workbench can be found at: http://apmd.prism.uvsq.fr/SubProject4/TestPlatform/ ([10]).

5. REFERENCES

[1] GroupLens Research: "movielens: helping you to find the right movies". Web site, ULR: http://movielens.umn.edu, last accessed on July 9th, 2007.

[2] Halevy, A., Pottinger, R.: "MiniCon: A scalable algorithm for answering queries using views", Very Large Data Bases Journal, Vol. 10, p. 182-198, 2001.

[3] Kostadinov, D.: "Data Personalization: an Approach for Profile Management and Query Reformulation", PhD thesis, University of Versailles, France, 2007.

[4] Koutrika, G., Ioannidis, Y. E.,: "Personalization of Queries in Database Systems", In Proc. of the 20th Int. Conference on Data Engineering, Boston, USA, p. 597-608, 2004.

[5] Internet Movie Database, Inc.: "The Internet Movie Database", Web site, URL: http://www.imdb.com/, last accessed on July 9th, 2007.

[6] Peralta, V.: "Extraction and Integration of MovieLens and IMDb Data". Technical Report, Laboratoire PRiSM, Université de Versailles, France, July 2007.

[7] Peralta, V.: "Generation of a Reference Data Set for Query Personalization". Technical Report, Laboratoire PRiSM, Université de Versailles, France, October 2007.

[8] Text REtrival Conference (TREC). URL: http://trec.nist.gov/, last accessed on September 2007.

[9] Transaction Processing Performance Council. URL: http://www.tpc.org/, last accessed on September 2007.

[10] URL of the APMD-Workbench: A Benchmark for Query Personalization Systems: http://apmd.prism.uvsq.fr/SubProject4/TestPlatform/.