Journal of Engineering Science and Technology
Vol. 12, No. 4 (2017) 1001 - 1021
© School of Engineering, Taylor's University

PDMS: AN EFFECTIVE PREFERENCE COUPLED DATA MANAGEMENT AND RETRIEVAL SYSTEM

SHAILESH P. KHAPRE[1,*], M. S. SALEEM BASHA[2],
SUJATHA POTHULA[1], P. DHAVACHELVAN[1]

[1]Department Of Computer Science, Pondicherry University Pondicherry, India
[2]Department of Computer Science and IT, Mazoon University College, Muscat, Oman
*Corresponding Author: shaileshkhaprerkl@gmail.com

Abstract

The exponential growth of the Web has turned it into a huge repository of information. Learning and extracting relevant information satisfying the consumers (User) needs is the ongoing research areas in Information retrieval. It is identified from the information retrieval study that stipulated focuses have been initiated for storing users data. Peoples mainly fail to look at ground level details, i.e., Retrieval of relevant data mainly depends on how efficiency a Data Management Scheme is applied. Data management plays an important role in effective and responsive handling of data. This paper shows the method for incorporating preferences in traditional Data Management schemes and how queries can be remodelled for confident data extraction; in this paper a prototype "Preference Data Management System" has been developed and implemented with proposed query optimization Algorithm Group Bottom Up. The experimental results show that the newly execution plan generated using both the greedy algorithm and by dynamic programming performs quite well compared to the optimal plan.

Keywords: Personalization, Preference mining, Query optimization. Information retrieval.

1. Introduction

In recent years the development of Commercial Personalization systems has been rapid. The Personalization was a key component of a wide variety of functional applications such as social networks, e-commerce, multimedia, news websites, advertising platforms, etc.

The most compelling reason that led to the development of personalization has to do with the amount of information and the range of options that are currently users of the various systems and applications, the flow of information and the variety of services offered especially through the internet goes much that the average person can manage and consume, (e.g., information, entertainment, communication, product, etc.) there are thousands of different options competing for the interest and attention of the users . This creates significant difficulties for users to choose that which best suits your interests and preferences. Furthermore, this oversupply of data and services is a major challenge for content providers trying to improve the user experience, to meet different needs and potential to attract more consumers. Therefore, the need for systems that perceive different tastes, needs and interests of the users has now gained immense importance both for consumers / users of such systems and to the service providers.

A second important reason that facilitated the development of personalization systems is that the collection, recording and processing of huge amounts of data on the characteristics, interests and preferences of users for the current user interface has become quite easier. Two main ways that such data is collected is through social networks and through internet access from mobile devices. In both cases the users themselves are willing to provide these data to the respective applications either directly (by creating an account / user profile) or indirectly by the use of the application. The data used for personalization include user demographics (age, gender, place of residence, etc.), preferences declared by users in response to questions or grading interesting information, or data collected by monitoring the behaviour, for example which pages visited, how long he stayed in cover, pages, comments or assessments made, etc. Very often the integration of information related to their interests and preferences of a user is very simple, or it can be done even without the direct involvement of the user. For example, many users of social networks share information of interest to them (e.g., music, news, photos and videos) or your current location or status.

Finally, another important factor that contributed to the development of personalization systems has to do with the kind of services offered via the internet and mobile devices (smartphones, tablets). While initially developed services mainly dealing transactions, today the internet has become a more social status. Indeed it is worth noting that most of the popular applications used daily by billions of people are in such a form. As an example we can mention the social web applications (e.g., Facebook, Twitter, LinkedIn), multimedia applications (e.g., YouTube, Flickr, Pandora, last.fm, Netflix), search local points of interest (e.g., Foursquare, Yelp), daily deals (e.g., Groupon, LivingSocial), search or recommendations or blogs (e.g., StumbleUpon, Delicious, reddit), electronic purchases and sales (e.g., Amazon, eBay), etc.

Research topics in personalization

In practice, the problem of personalization is quite complex. Therefore, in parallel with the development of commercial systems, personalization issues attracted the keen interest of the research community. Many different disciplines are dealing with this subject, such as the social sciences (e.g. psychology), the Machine Learning, Data Mining, Artificial Intelligence, databases, Information Retrieval and Human Computer Interaction. Each sector has a significant contribution to personalization system. Psychology, machine learning, data mining and artificial intelligence mainly focus on collecting preferences, export rules and creating user profiles and behavioural patterns. Databases and Information Retrieval dealing with data management systems that depend on the preferences and query valuation techniques with preferences for structured and unstructured data respectively, The Human-Computer Interaction focuses on the adaptation of the interface in the preferences and user interface.

Focusing on Databases research issues arising towards personalization are quite interesting. For example some problems remain open as the generalized representation of preferences for structured data, proper indexing of data and preferences, selecting appropriate preferences for a query, how their integration in a query, the method of selecting the most relevant results , ranking algorithms, etc.

This paper emphasizes on Personalized Data Management. It is specifically focusing on: (a) Relational Data Management Systems relying on user preferences, and (b) Preference Query valuation.

In Section 2 an introduction of the proposed Representation model for Preferences, the extended data model and queries has been discussed; Section 3 contains a summary of the system Preference Data Management System (PDMS). Section 4 deals with the optimization and execution of queries in the system preferences PDMS. Section 5 presents the experimental evaluation of the system.

2. Model Representation of Preferences

Consider a relational database. Every relationship R_B consists of a set of attributes $A = \{A_i, \ldots, A_d\}$ and let us consider a record t belonging to a instance of the relationship R_B, For each attribute A_i; $1 \leq i \leq d$, where the notation $t.A_i$ denotes the value of t for attribute A_i; and dom(A_i) for range of attribute A_i.

A preference P is defined on a relation R_B as a triplet consisting of (a) a preference condition (condition) by setting the records affected by preference, (b) a function scoring (scoring function) used to rating of these recordings, and (c) a degree of confidence (confidence) that determines how strong and confident is the specific preference. It is explained in more details as follows:

Definition1:Preference. A preference p in relation R_B is defined over as a triplet(σ_ϕ, S, C), where σ_ϕ represents a preferred condition involving a set of attributes $A_\phi \subseteq A$ on relation R_B, S is a scoring function defined on the Cartesian product of a set of attributes $A_s \subseteq A$ of R_B, so S:$\prod_{A_i \in A_s} dom(A_i) \rightarrow [0,1] \cup \{\perp\}$, and C is a constant with interval [0,1].

In other words, a preference p assigns each record t belonging to $\sigma_\phi(R_B)$ a numeric score by applying the scoring function S with certainty C. Following this semantics, a profile t is preferred to record a t', where t is the highest score from t'. The price $\perp$ is used as the default score to indicate no preference for any particular record. Also the value of 1 for a score expresses absolute preference, while a value of 0 expressing total dislike for the specific record.

It should be noted that the ranges for the score and the degree of certainty could be different, for example use of positive scores for preference match and negative scores for repulsion, and the choice of range is orthogonal to the proposed framework. An interval $[0,1] \cup \{\perp\}$ for the score has been chosen, because in most applications users are more accustomed to score objects in a positive rather than a negative scale , for example, by giving 1-5 stars or ratings between 1 to 10, where smaller values indicate dislike.

The scoring function S is possible to assign the same fixed score in all the records in $\sigma_{\phi}(R_B)$ or assign different scores to each record based on the values for the attributes A_s, in the literature several methods for the extraction of a scoring function S have been proposed so that it reflects the preferences of a user as far as possible, example using machine learning techniques [1-5], extracting preferences based on previous (query log mining [6], recommender systems [7] the decision-making multiple criteria [8-10], Context mining [11-13], etc.

The degree of Confidence reflects the uncertainty created by the method of learning the preferences of a user. Intuitively, greater confidence indicates stronger indication of a preference. For example, a preference is declared explicitly by the user himself will have the greatest degree of confidence that is equal to 1 in contrast with preferences that are not expressed by users but have been exported from the system, the degree of confidence depends on the followed method of learning, and is lower than 1, for example, if a user has watched many comedies, then a learning method can infer that the user loves comedies. In this case, the degree of confidence that will be assigned to specific preference should take into account the sample size, i.e., the number of comedies that user watched monitoring in relation to the total number of movies in the database. The degree of confidence is used as a preference weight that affects the overall score of a record.

2.1. Expanded relational model

Aiming to enrich the records in a database with additional values for the score and the corresponding degree of confidence, an expanded relationship can be defined as:

Definition 2: Expanded Relationship. Given a basic relationship R_B $(A_1, \ldots, A_d$, an expanded relationship R is defined by extending the basic relationship R_B with (a) a condition scores range S with $dom(S) = \Re^{+} \cup \{\perp\}$, and (b) a condition confidence C with range $dom(C) = \Re^{+}$. The default value for the score is ± and the confidence level is 0, i.e., $R = \{(t, S_t, C_t) | t \in R_B, S_t = \perp, C_t = 0\}$. Each entry t associated with a unique pair of values $\langle St, Ct \rangle$.

Note that although the maximum value for the score and the degree of confidence that can be assigned to a preference is 1, a record may have higher scores or values of trust as a result of the combination of several scores or degrees of trust applicable to specific record.

There are two ways in which a pair of scores and degree of confidence can be assigned to a record t. The first way is by evaluating a preference on the relationship. The second way is by transferring such contribution values from other relationships with the help of an operator. In the first case, when evaluating a preference in the relationship, it can record t already have a confidence score. Therefore, there is a need to combine the old pair scores and degree of confidence with the new one and assign the final pair in t. In the second case, when a new record is formed as a result of a relational operator, the values for the score and the confidence is generated by the scores and degrees of confidence of the original records. For example, when evaluating an operator coupling (join), the pair of values-confidence degree score that will be assigned to the generated record will be same as values of the entries of original relationships.

For both cases, the combination of two pairs of scoring and confidence degree is achieved by means of an aggregate function. According to the proposed model, a score is always connected with a degree of confidence. Hereinafter a pair of score-degree of confidence will be denoted as $\langle S, C\rangle$, it is also obvious that an aggregate function must be able to handle pairs instead of individual values of score and degree of confidence. Below is the definition of an aggregate function.

Definition3: Aggregate function. An aggregate function $F: \langle S, C\rangle X\langle S, C\rangle \to \langle S, C\rangle$ combines two pairs of score-degree of confidence in a final pair of values.

Preferences are indicated using binary comparisons between attribute values of a relationship [14-17]. In this case there may be values that are not comparable to each other and therefore resulting partial order of the records.

Intuitively, a change in the order valuation of preferences should not affect the final score-degree of confidence pair. Therefore, there is a need to apply the associative and commutative properties for aggregate functions. Moreover, the congregation should satisfy the following properties:

$F: (\langle \perp, 0\rangle, \langle \perp, 0\rangle) = \langle \perp, 0\rangle F: \langle \perp, C\rangle X\langle S, C\rangle = \langle S, C\rangle$

The aggregate functions can be used to implement different viewpoints or strategies on how to combine the preferences in an application, the selection of appropriate aggregate function in each case depends on what attitude more accurately reflect the way the system should behaves in practice.

2.2. Expanded basic operators of relational algebra

The traditional relational algebra operators have been extended so to be able to handle the expanded relationships.

Selection operator$\sigma_\phi(R_B)$, selects those records from an expanded relation *R* which satisfy the condition ϕ, i.e., $\sigma_\phi(R_B) = \{t | t \in R \wedge \sigma_\phi(t)\}$. The selection condition can also implicate new features scores and degree of confidence.

Projection operator, $\pi_{A_1, A_2, \ldots, A_k}(R)$, relies on a subset of features of the expanded relationship *R*, i.e., $\pi_{A_1, A_2, \ldots, A_k}(R) = \{(t.A_1, \ldots, t.A_k, S_t, C_t) | t \in R\}$

In addition to the projected attributes, the projection operator always maintains the attributes score and degree of confidence, so that the final formula derived to be expanded.

Intercept operator, $R_i \cap_F R_j$, takes as input two expanded relations R_i and R_j and produces an expanded relationship which includes all records belonging both R_i and R_j with the scores and degrees of confidence generated by combining the scores and degrees of confidence of records that are in the initial relationship R_i and R_j, i.e., $R_i \cap_F R_j = \{(t, S_t, C_t) | (t, S_i, C_i) \epsilon R_i \wedge (t, S_j, C_j) \epsilon R_j, \langle S_t, C_t\rangle := F_S(\langle S_i, C_i\rangle, \langle S_j, C_j\rangle)\}$.

Union operator, $R_i \cup_F R_j$, takes as input two expanded relations R_i and R_j and produces an expanded relationship which includes all records belonging either R_i. R_j either or both (double entries omitted) with a final score and degrees of confidence generated by combining the scores and degrees of confidence of records placed in the original relations R_i and R_j, i.e.,

$$R_i \cup_F R_j = \{(t,\ S_t, C_t) | (t,\ S_i, C_i) \epsilon\, R_i\ \wedge\ (t,*,*) \notin\ R_j, \langle S_t, C_t \rangle := \langle S_i, C_i \rangle\ \vee$$

$$\left((t,\ S_j, C_j)\ \epsilon\, R_j\ \wedge\ (t,*,*) \notin\ R_i, \langle S_t, C_t \rangle := \langle S_j, C_j \rangle\right) \vee$$

$$((t,\ S_i, C_i)\ \epsilon\, R_i\ \wedge\ (t,\ S_j, C_j)\ \epsilon\, R_j, \langle S_t, C_t \rangle := F_S\ (\langle S_i, C_i \rangle, \langle S_j, C_j \rangle))\ \}$$

Coupling Operator, $R_i X_{\phi,F} R_j$ takes as input two expanded relations R_i and R_j and produces an expanded relationship as

$$R_i X_{\phi,F} R_j = \{(t, S_t, C_t) | t = t_i\, X_\phi\, t_j, (t_i, S_i, C_i)\ \epsilon\, R_i, (t_j, S_j, C_j)\ \epsilon\, R_j$$

$$\langle S_t, C_t \rangle = F(\langle S_i, C_i \rangle, \langle S_j, C_j \rangle)\}$$

Preference Operator, this section deals with the extension of relational algebra with a new operator preference $\lambda_{p,F}(R)$.The operator preference evaluates a preference p expanded in a relationship R using the predetermined function F aggregation to combine the previous score-degrees of confidence with new results from the current preference. More specifically, the operator preference$\lambda_{p,F}(R)$ executes a preference $p :=\ (\sigma_\phi, S, C)$ in an expanded relationship R and generates a new expanded connection as follows:

$$\lambda_{p,F}(R) = \{(t, S'_t, C'_t) | (t, S_t, C_t) \in R\}\ and\ \langle S'_t, C'_t \rangle = \begin{cases} F(\langle S_t, C_t \rangle, \langle S(t), C \rangle), & if\ t \in\ \sigma_\phi(R) \\ \langle S_t, C_t \rangle, & else \end{cases} \}$$

3. Preference Database Management System (PDMS)

PDMS is a prototype system implemented and expanded based on the relational model and query model.

Figure 1 illustrates the system architecture of PDMS. The subsystems are shown in yellow provided by the database; while blue colour distinguished subsystems developed for the system PDMS. As the figure shows, the PDMS provides two options for introducing queries preferences. More specifically, a user's preferences can be specified along with the question or the system can automatically enrich a conventional query related preferences Following the first choice, preferences are given a noticeable way above the rest of the query. In the second case the relative preferences provided by the Preferences Management Subsystem (profile manager module), which accesses preferences already collected from the database.

Saved preferences may have collected from previous questions preference set by the user or by analysing the validations given by the user in the past, or the behaviour of the system. Note that the method of collecting and learning preferences of a user are orthogonal to the processing of a query preference, which the primary functionality is provided by the system PDMS. Therefore, for the purposes of the original system implementation, there is need to store (a) preferences declared by the users themselves through a graphical interface that have developed and (b) preferences declared in previous user queries .

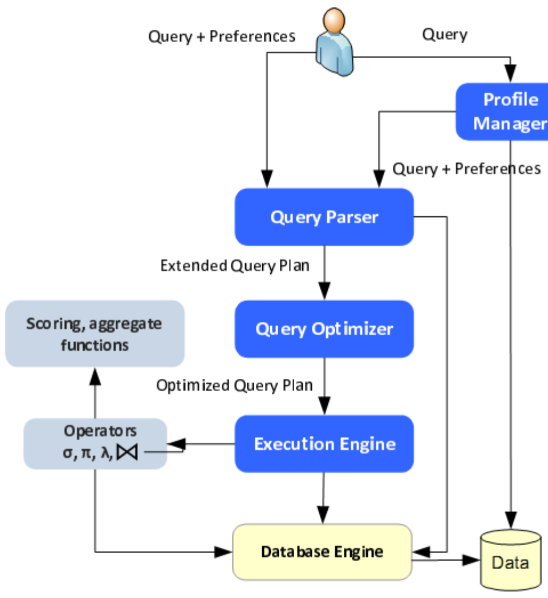

Fig. 1. System architecture of PDMS.

Regardless of how preferences are given as input to the system, the query with preferences is indorsed in the Query Analyser Subsystem (*Query Parser*).

Below is an overview of key subsystems of the system *PDMS*,

- The management module "*Profile Manager*" chooses the preferences that can be combined with the conditions of the query. For this reason, the algorithm preferences selection proposed in [18] has been used.
- The analysis module "query parser" takes as input a query and a set of preferences and manufactures an extended execution plan which is promoted for module "*Query Optimizer*".
- The module "*Query Optimizer*" improves the execution plan that takes as input by applying a set of heuristic rules based on properties of preference operator. The improved plan is used as the basis for the next optimization step that follows the proposed cost mode. The objective of optimization is to choose between several alternative plans, one with the minimum cost of implementation.
- The subsystem "execution engine" takes as input the selected execution plan and executes it by following one of the suggested methods of execution.

3.1. Implementation of the system

Prices for the score and the degrees of confidence of the extended relationship are calculated by evaluating a query with preferences. Moreover, usually the most of the records remain unaffected by the preferences indicated with a query. Therefore, it makes no sense for permanent storage of scores and trust in database. Instead following strategy has been devised.

For each base table R_B involved in a query with preferences produces a corresponding scoreboard $R_S\{p_k,score,conf)$, where p_k denotes the possibly composite key of the base table R_B. To save additional space, table score only contain records with non-zero score, i.e., those affected by the preferences of the query. Therefore, as expected in this case $|R_s| \ll |R_B|$. Table R_S is updated with new values for the scores and confidence whenever an operator is performed on the specific relationship.

Implementation Operators

All Expanded operators that involve in relationships that do not contain records with non-zero score, can be transformed into classical relational operators. Before carrying out any operation, the system checks if the PDMS corresponding scoreboard is empty. If it is empty, execution is performed with the operator assigned to the underlying execution engine. Otherwise, the system makes use PDMS implementations of operators as described below.

If any expanded relationship as a pair of a key and a score table is implemented then each operator must be performed in both tables. For example, when running a selection operator, the records that do not satisfy these conditions are rejected by both the tables. Operator's projectors are simpler: all the features of the scoring table must necessarily be included in the view. Operands coupling and the theoretic operators performed in two steps. First, a conventional coupling (respectively union or intersection) performed at two base tables. Then, for the records involved in the output relation, scores and confidence is calculated by applying the corresponding aggregate function to the original tables score.

The execution of an operator preference is somewhat more complicated. Specifically, the initial condition that is preferably performed in both tables R_B and R_S. All table entries that satisfy condition preference already and having nonzero scores and degrees of confidence, therefore should be assigned new values. In addition, the records of R_B base table that do not appear in resulted scores of R_S, should be calculated. For the calculation of scores and confidence level for both categories documents ranking function of preference applies to records that satisfy the condition preference. Lastly the corresponding aggregate function is called to calculate total prices.

3.2. Query processing in the system PDMS

Queries Analysis. Given a query and a set of preferences, queries analyser module initially constructs an extended execution plan which contains all expanded operands and preference operator. In the generated execution plan, the number of operators follows the order of their appearance in the original query. The analysis subsystem query adds a preference operator for every perception. Also, check each preference whether a feature (either in regard to preference or scoring function) which is not contained in the query. In this case the projection operator suitably modifies to include these features. Furthermore, the queries analyser module, adds additional coupling conditions with relationships that may not appear in the original SQL query, but are involved in preferences declared along with the query. For example, preference is given to relationship *ACTORS*, and therefore analysis subsystem queries will add the following coupling conditions *MOVIES X CAST X ACTORS*. Figure 2 shows the execution plan generated for the example above question.

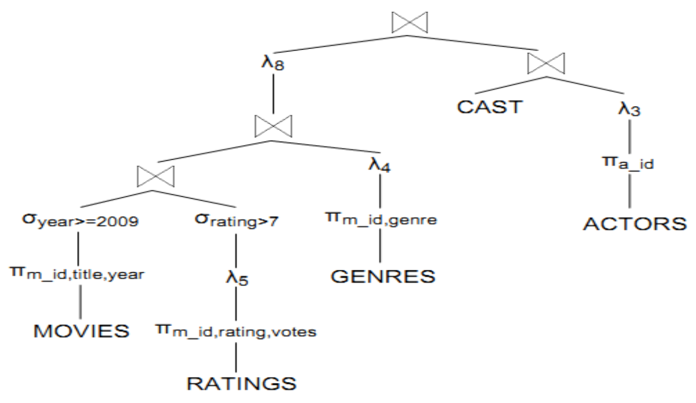

Fig. 2. Execution plan.

Query Optimization. As described above, an extended execution plan includes expanded operands and preference operators. It is thus evident that a conventional query optimizer of a database cannot be used for such execution plans. Nevertheless, the implementation of expanded relationships following the PDMS system separates the records related to the preference valuation (those contained in Tables score) with those associated with the remaining part of the query (those contained in database tables). Therefore, the total cost of implementation consists of two parts, a piece associated with the calculation and aggregation of scores and confidence score tables, and a piece associated with other operators (selection, couplings, etc.) database tables.

The expanded relational operators do not change the way the records are filtered based on the conventional definitions. Also, the performance of a preference operator does not cause the rejection of any recording. Therefore, the new operators do not affect the part of the processing costs associated with the base tables. It is therefore expected that the order of execution joins would select the query optimizer of the database for a query if the operands are removed and will perform well for the same query with preferences. Based on this observation, the query optimizer PDMS system maintains the execution order joins selected from the database, and considers this piece of execution cost as stable. Therefore, the target for optimizer is to minimize the remaining costs, i.e., the costs associated with the measurement preferences.

Usually the most critical parameter affecting the cost of processing a query is the number of I/O operations from the disk, which is proportional to the number of records. Making the assumption that other operators have a fixed position in the execution plan, the main goal of the optimizer is to place the preference operators in the execution plan, so the number of records that are transferred through tables score is minimized.

Based on the above, the optimizer strategy includes the following steps. Export the original sequence for joins without preferences. Then, using the algebraic properties the optimizer applies a set of heuristic rules that are intended to improve the position of the operators preference in execution plan, the goal of this step is: (a) attempt to minimize the number of records that are given as input for preference operators, and (b) reject less efficient (suboptimal) plans and thereby restrict the search space of alternative plans that will look to the next step. Finally, two algorithms have been proposed for cost estimation which (a) consider alternative positions of the preference operators, and (b) choose the plan with the least estimated cost.

(i). ***Rules based query optimization:*** Initially, the optimizer retrieves the execution plan form database. Then the optimizer rearranges the plan received as input from the query analyser, so as to follow the recommended sequence for joins. Finally, the modified plan is produced by applying a set of equivalence rules exploiting algebraic properties of preference operator. The objective is to minimize the number of records wherever possible. Specifically, the query optimizer applies the following heuristic rules:

(1). Selection operators involving attributes except scores and degrees of confidence are pushed lower as much as possible for the execution plan. If a selection condition involves more relationship, breaks the condition and move each piece lower down separately.

(2). The projection operators are pushed as low as possible in the execution plan, but ensuring that all the required attributes will be available next operands.

(3). If a preference operator is positioned higher than a binary operator (e.g., coupling, union, intersection) and simultaneously involves attributes from only one of the two relations, then push lower down from the corresponding binary operator relationship.

(4). If an preference operator and selection operator is defined over the same relation, then all conditions of options involving attributes outside of the score and the degree of confidence are added to the condition of preference operator.

The heuristic rules 1 and 2 are widely used for the optimization of conventional execution plan, aimed at reducing the number and size of records that flow through the execution plan respectively. The heuristic rule 3 is applied so that each operator preference be placed at the lowest position of the plan execution,

Figure 3 shows the execution plan generated after applying heuristic rules in plan in Fig. 2. It can be observed that the operator λ_5 has changed position with the operator rating > 7, while the operator λ_8 has transferred below the coupling operator, since it involves only attributes from the left part of the operator (*MOVIES and GENRES*).

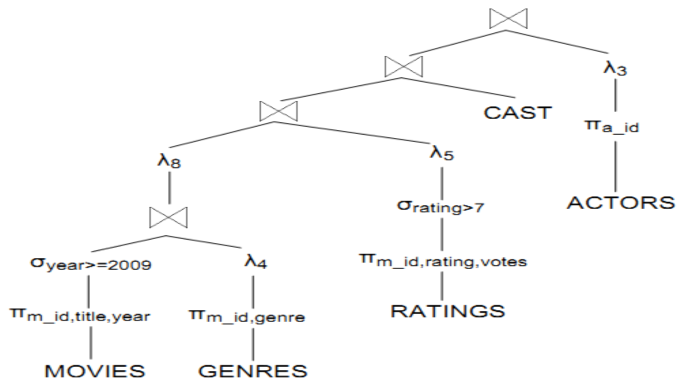

Fig. 3. Execution plan after applying heuristic rules.

(ii). ***Query optimization based on cost:*** This section deals with introduction of a cost model for individual operators, and how to estimate the total cost of an execution plan based on the positions of the preference operators. Finally, two algorithms based on cost estimation have been proposed that choose the most efficient execution plan.

Cost model. Consider the cost of a preference operator is proportional to the number of records affected by the operator, following an approach similar to that followed for the estimation of the cost of conventional operators of the relational algebra. Let cost (R_{B_i}, λ_j) the cost of a valuation of preference operator λ_j on base table R_{B_i}. Then we get Eq. (1):

$$cost\ (R_{B_i}, \lambda_j) \cong sel(R_{B_i}, \lambda_j).|R_{B_i}| + \alpha.sel(R_{S_i}, \lambda_j).|R_{B_i} \quad (1)$$

where R_{B_i} and R_{S_i} symbolize base tables and score and $sel(R_{B_i}, \lambda_j)$ and $sel(R_{S_i}, \lambda_j)$ denote the selectivity of the operator λ_j when applied to tables R_{B_i} and R_{S_i} respectively. Note that the cost of calculating new scores and degrees of confidence (which is related to the records contained in the database table) is different from the cost of gathering score and confidence level (which relates to the records contained in the index score), so for this reason a normalization factor is used in Eq. (1) when both above costs are added.

To estimate the cost of the union, intersection and coupling operators, the size of the database table that is expected to produce is used, according to the statistical database. Here the focus is on the cost of gathering score and confidence degrees. Because the distribution of records in these tables depends on the preferences that have been measured in the previous steps of the query execution, Using the following estimates corresponding to the worst case:

$$cost\ (R_i \cup \mathrm{R}_j) \cong |R_{S_i}| + |R_{S_j}| \quad (2)$$

$$cost\ (R_i \cap \mathrm{R}_j) \cong \min\{|R_{S_i}|, |R_{S_j}|\} \quad (3)$$

$$cost\ (R_i X\, \mathrm{R}_j) \cong sel\left(R_{B_i}, R_{B_j}\right) |R_{S_i}|.|R_{S_j}| \quad (4)$$

where $sel\left(R_{B_i}, R_{B_j}\right)$is the selectivity coupled table R_{B_i}, R_{B_j}.

As explained above, an accurate cost estimate should take into account: (a) the cost associated with the tables score, and (b) the cost associated with conventional piece of executing a query (e.g., selection, projection, coupling base tables). However, since the execution order of joins are predetermined and same for all implementation, there is need to examine the costs associated with their base tables. Therefore, the cost of an execution plan *plan*$_q$ is associated with the valuation of preferences, i.e.:

$$cost\ (\mathrm{plan}_q) \cong \sum_{p\_op \in plan_q} cost(p\_op) \quad (5)$$

where *p_op* denote an operator in execution plan.

(iii). ***Estimated cost:*** Figure 3 shows the plan produced by the first optimization step where for simplicity in our presentation the projection operators and preference operators have been removed. Let $\boldsymbol{\lambda_j}$ an preference operator assesses a preference $\boldsymbol{p_j[R]}$, According to the heuristic rule 1, the operator $\boldsymbol{\lambda_j}$ should have been placed higher than selection or projection operator, if any. Based on the algebraic properties of operator preference and making the assumption of the appropriate projections on the required attributes, operator $\boldsymbol{\lambda_j}$ can be chosen to apply at any position between the current and the root of the execution plan. Alternative positions for the operator $\boldsymbol{\lambda_5}$ is shown circled in Fig. 4.

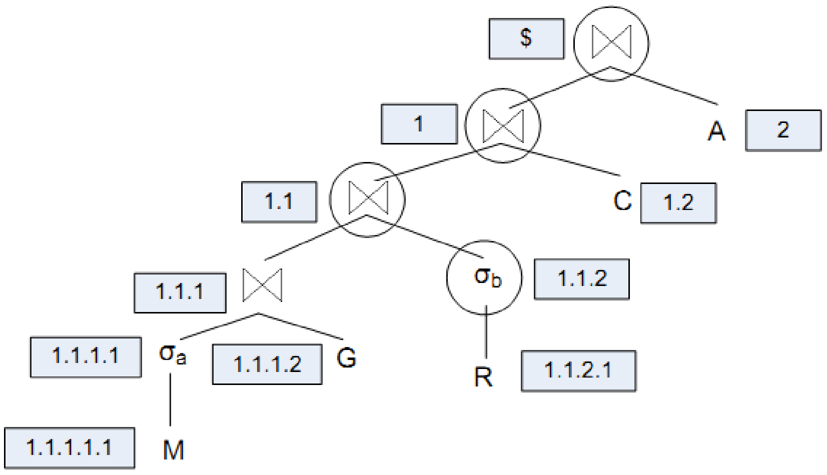

Fig. 4. Alternative positions for operator, λ_5.

Considering a predefined execution order joins, the search space consists of all the instances that follow this sequence for joins, but differ in the exact position of the preference operators. Therefore, the optimization task is to find an execution plan where the preference operator is positioned in such a way as to minimize the above cost ($plan_q$).

Let h the number of plan execution and let p be the number of preference operators included in plan. An exhaustive examination of all alternative plans would require $O(h^p)$ cost estimates, which is prohibitively expensive in terms of time even for small values of h and P. It should be noted of course that because of the first step optimization based heuristic rules; the size of the search space has already reduced, as they have rejected some instances that are less efficient. For example, Returning in Fig. 4, the second step in the cost-based optimizer will not consider instances in which the operator has applied directly over the relation R, as the operator of choice has moved lower and precedes preference operator, λ_5.

Approximation Algorithms. Two approximation algorithms have been proposed for cost estimation that leads to an efficient execution plan by examining a subset of alternative plans.

(i). *Dynamic programing:* The algorithm starts by considering all possible positions for a pair of preference operators. Calculate the cost of each plan and each operator preference and mark the position which minimizes the estimated cost. In the *i*-th execution step, the algorithm examines all subsets of preference operator size *i* for each combination and notes all the best places. In the next step, all combinations generated effector sized i+1 subsets expanding size *i* with a further operator preference. For each subset of operators that have already been examined using the best positions as calculated in any of the previous steps, the algorithm terminates when completed cost estimate and P operators preference, so returned all the best places.

It should be noted that any cost estimate requires the identification of all sets that are higher from the current position of an operator, in order to establish the valid locations and update the expected costs appropriately, this can be done most efficiently by using an encoding scheme for trees, i.e., tree labelling scheme. Specifically, the coding used Dewey, which allows the query to find response-ancestor nodes (ancestor queries) in constant time. In Fig. 4, each node is coded by Dewey. For example, nodes 1 and 1.1 are alternative places for the operator λ_5.

(ii). ***Greedy algorithm:*** The greedy algorithm Sequence the preference operators in execution plan, considering the ascending order of minimum cost. At each step, the greedy algorithm chooses between preference operator, whose location is yet to be determined and has the lowest cost of implementation. The algorithm notes this location as optimal for the current test preference operator, and informs appropriate costs of implementation of all operators that are positioned higher in the execution plan. The algorithm is performed in iterations until the position of all of the preference operators has been determined, consequently modifying the execution plan in order to reflect the new positions of the selected effector preference.

Basic approach for handling an expanded execution plan would be to perform a post-order traversal of the execution plan and implement each operator following the execution order, this algorithm is referred as Bottom-Up (*BU*).The *BU* algorithm traverses the plan chosen by the query optimizer and performs each operator separately. For each intermediate node execution plan n_i, algorithm *BU* holds the results of the execution of this operator to a pair of temporary tables $T_{B_i}$and T_{S_i}. The *BU* algorithm terminates when executed and operand is placed in the root of the plan execution. The final query results are obtained by performing a join between tables T_{B_r} and T_{S_r}.

It is obvious that the *BU* algorithm is inefficient as it takes all intermediate tables to be generated to write to disk during the execution of the query. Therefore, based on the observations made above, a more efficient algorithm, Group Bottom-Up (*GBU*) has been proposed which: (a) combines operator's execution attempts to avoid few disk writes, and (b) uses the query execution module for individual parts of the execution of a query. By combining some operators, all the intermediate tables writing to disk can be avoided. Furthermore, for example by combining two or more preference operators, the corresponding stored procedure which implements the preference operator is called only once. Also, transferring some parts of the query in the database helped in exploiting some optimizations, such as efficient data access methods, efficient implementation of operators and pipelining between operators. In order to solve our original objective, namely to minimize the number of records to disk, a set of rules has been identified that the *GBU* algorithm examines in order to decide whether an operator can be combined with following operators:

- The execution of an operator selection or projection operator may be postponed.
- Successive preference operators can be performed in one step.
- Execution of a coupling operator can be deferred if at least one of the two original tables is empty.

The *GBU* algorithm follows a post-order traversal for expanded plan execution. It traverses, rather than directly execute each operator n_i, *GBU* algorithm examines whether the implementation of n may be deferred in accordance with the rules quoted above. To execute groups of operators, the algorithm *GBU* maintains a directed acyclic graph (*DAG*) *G* containing intermediate tables.

Algorithm 1: Group Bottom Up

```
Input: $Q_p$ a query plan, $n_r$ the root of $Q_p$
Output: $R_Q = \{(t, S_t, C_t) | t \in R_{NP}\}$ where $R_{NP}$ is the result of executing the non-preference part of Q.
Begin
$G = \phi$;
$GBU(n_r, G)$;
$extract(G, n_r) \rightarrow q'$; //Remove remaining operators, combine extracted operators into q'
$extract(q') \rightarrow T_{B_r}, T_{S_r}$;
$R_Q = T_{B_r} x\, T_{S_r}$;
$return\ R_Q$;
```

To execute groups of operators, the algorithm *GBU* maintains a directed acyclic graph (*DAG*) *G* containing intermediate tables generated during the execution of the plan that have not yet been used, and those operators whose execution has been postponed. Whenever the algorithm *GBU* accesses a node *n* whose execution can be postponed, then the n_i is enrolled in *G* (lines 6, 8, 12 and 22 in function *GBU*). If the algorithm visits an operator who must be executed immediately, then all nodes belonging to the subtree of node *n* is deleted from the graph *G* and combined into one sub-query (lines 15 and 24-25), which is then sent to the database queries executing module (line 16). The results of the entered query are stored in a pair of temporary tables and then given as input for operator *n* (lines 17 and 26). Intermediate results produced by executing the operator *n* are inserted again in the graph *G*. When the algorithm *GBU* reaches the root of the execution plan, then all operands which are left in the graph *G* are combined and executed in one step (lines 4 to 5 in Algorithm.1), finally executing a join between tables T_{B_r}, T_{S_r} that produced as a result of the execution of the operator is positioned at the root of the plan execution (line 6) and returns the final results(line 7).

```
Function GBU
 Input: $n_i$, node in Qp, G a DAG containing all operators that can be
 combined and intermediate tables produced during query execution.
Output: G
1.  begin
2.  if $n_i$ is null then
3.  exit;
4.  GBU($n_i$ .left, G); GBU($n_i$ .right, G);
5.  if $n_i$ is a relation (leaf node) $R_i$ then
6.          insert $R_i$ into G;
7.  else if $n_i$ is a project or select operator then
8.          insert $n_i$ into G;
9.  else if $n_i$ is a prefer operator then
10.         let $n_p$ be the parent node of $n_i$ ;
11.         if $n_p$ is a prefer operator then
12.              insert $n_i$ into G;
13.         else
14.                   let $n_j$ be the child node of $n_i$ ;
15.                   $extract(G, n_j) \rightarrow q'$; //Remove all not executed operators
                      in the subtree of $n_j$, combine extracted operators into q'
16.                   $execute(q') \rightarrow T_{B_j}, T_{S_j}$;
17.                   $evaluate(n_i, T_{B_j}, T_{S_j}) \rightarrow T_{S_i}$;
18.                   insert $T_{B_j}$ into G; insert $T_{S_i}$ into G;
19.  else if $n_i$ is a join or set operator then
20.         let $n_j$ , $n_k$ be the children nodes of $n_i$ ;
21.         if (($n_i$ is a join) and ($|T_{S_j}|$= 0 or $|T_{S_k}|$= 0)) then
22.                   insert $n_i$ into G;
23.         else
24.                   $extract(G, n_j) \rightarrow q_j$ ; //Remove all not executed operators
                      from both subtrees
25.                   $extract(G, n_k) \rightarrow q_k$;
26.                   $evaluate(n_i, q_j, q_k) \rightarrow T_{B_i}, T_{S_i}$; //combine all extracted
                      operators with $n_i$, execute query
27.                   insert $T_{B_i}$into G; insert $T_{S_i}$ into G;
```

4. Experimental Results and Evaluation

In this section, experimental evaluation has been carried out on the proposed methods. The aim of our experiments are (a) to evaluate the effect of the preferences valuation in the performance of executing a query using both the proposed methods of implementation, and plug-in methods, and (b) evaluate the proposed cost-based optimization algorithms.

Eight key query was used which try to select query with different characteristics compared to the number of results, the number of joins, the number of preferences, etc., in order to examine the behaviour of the methods in different usage scenarios, where $|Q|$ denote the number of query results without preferences, $|R|$ is the number of relations involved in the query and $|P|$ is the number of preferences. Preferences are uniformly distributed in the relations involved in the query. For the baseline experiments a preference per connection is used. For example, the query IMDB-3 involves three preferences, each one of which relates to a connection, while a connection is not associated with any preference.

```
PUBLICATION (p_id, title, pub_type), AUTHORS (a_id, name),
PUB_AUTHORS(p_id, a_id), CITATIONS (p1_id, p2_id),
CONFERENCES (p_id, name, year, location),
JOURNALS (p_id, name, year, volume)
```

Algorithm	*Average optimization time*	*(%) Improvement over PL*
GBU-G	123msecs	41.93%
GBU-D	234 msecs	44.44%
GBU-E	403 msecs	51.40%

The database DBLP is a snapshot of the database “Database of Scientific Publications” which was recovered in June 2015.

Performance in relation to the number of results:

In the first experiment, we measured the performance of all the methods in connection with the number of results |*Q*|. To isolate the number of results as a parameter, each executed query contains a join. Keeping one of the two parts and join using different relationships to each other, questions arise with different number of results. As in Fig. 5 shows the two variants of the algorithm *GBU* tested exhibit better scalability compared to methods plug-in *PL* and *FP* with respect to the results of a plurality question.

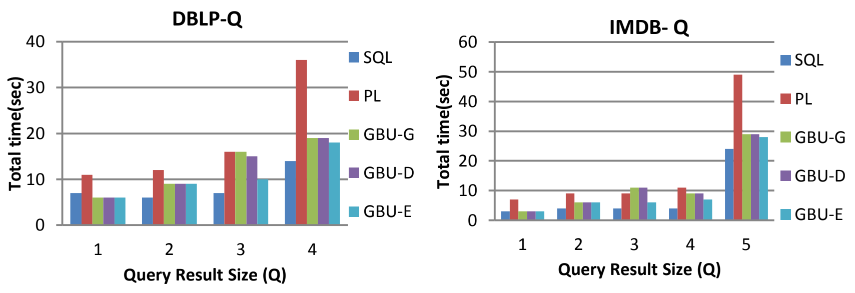

Fig. 5. Total execution time in relation to the number of results.

Performance in relation to the number of a query:

Then we compare all experimental methods with varying the number of tables |*R*| involved in a query. For the experiment, we used the following two questions IMDB-R1 with IR=3 and IMDB-R2 with IR=2 gradually adding additional tables to IR=6. Figure 6 illustrates the measured total execution times compared to the number of a query tables. As shown in the diagrams, both the *GBU-G* and the *GBU-D* is significantly faster and behave better scalability for queries with multiple joins compared with the plug-in methods we tested.

Performance in relation to the size of a query tables:

Figure 7 shows for queries where the size of the tables is relatively small, the plug-in methods have similar performance to that of the variants of the algorithm GBU- *. Conversely, as the size of the tables increases, the cost valuation preferences become appreciable factor in shaping the overall cost. In this case the execution algorithms that take into account the costs of enforcement are much more efficient than the methods of plug-in. Our experiments show that both proposed algorithms achieve optimization in most cases choose fairly efficient execution plan.

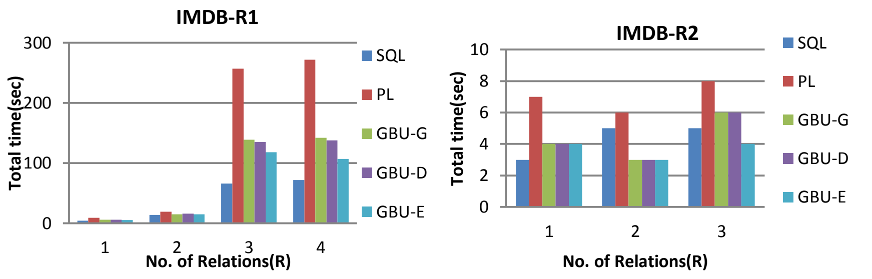

Fig. 6. Total execution time relative to the number of tables in the query.

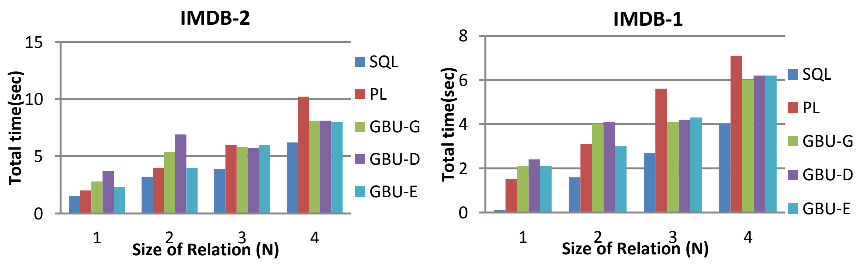

Fig. 7. Total execution time in relation to the size of query.

Performance in relation to the number of preferences:

In this experiment, we examine the performance of all methods varying the preferences number |*P*| associated with the query. As shown in Fig. 8, the algorithms GBU-G and GBU-D are more efficient and have better scalability ticked off as the number of preferences in a query. The gain in efficiency depends on the total number of query preferences. The more the preference operators need to evaluate the greater the gain in runtime methods are based on the algorithm GBU- * compared with the methods of plug-in.

In relation to the Query optimization, the dynamic programming algorithm and the exhaustive search algorithm is practically inapplicable for queries with a large number of preferences. According to our experiments as shown in Fig. 9, the exhaustive search cannot be applied for more than five preferences, while over 7 preferences dynamic programming algorithm implies higher total times compared with the greedy algorithm. Isolating execution times, the two algorithms GBU-G and GBU-D have similar performance to that of an algorithm that selects the best execution plan.

Performance relative to selectivity of preferences:

Then the effect of selective preferences on the performance of all methods is examined. For this experiment, we used queries IMDB-3, IMDB- 4, IMDB-5 and DBLP-3, wherein the condition changes of preference are more less selective. Figure 10 shows the results of each experiment. As expected, the performance of all methods deteriorates when preferences affect more recordings Nevertheless, variants of the algorithm GBU is in most cases more efficient, regardless of selectivity.

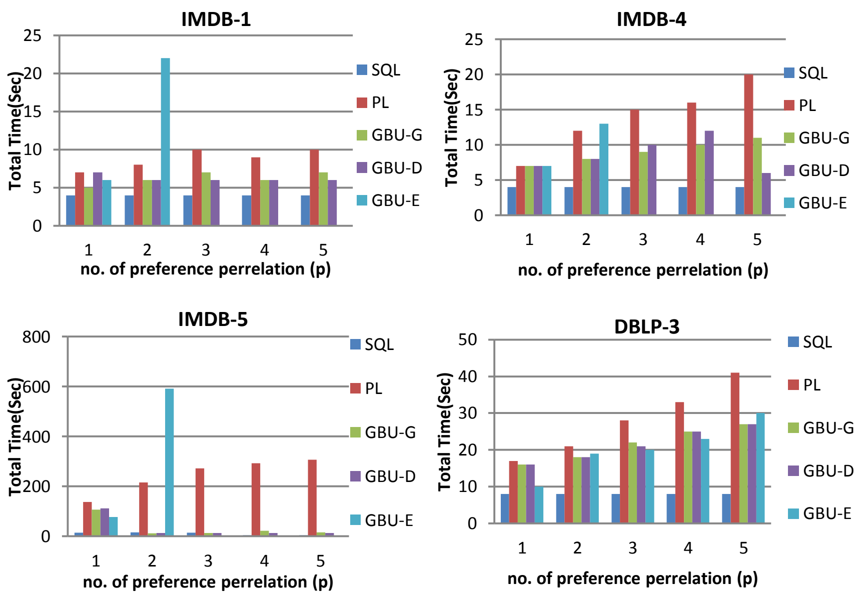

Fig. 8. Total execution time in relation to the number of preferences.

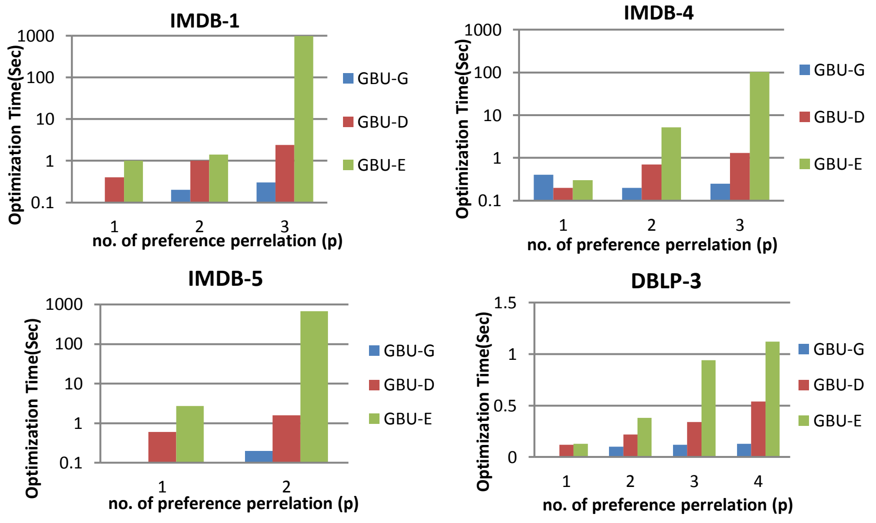

Fig. 9. Query optimization in relation to the number of preferences.

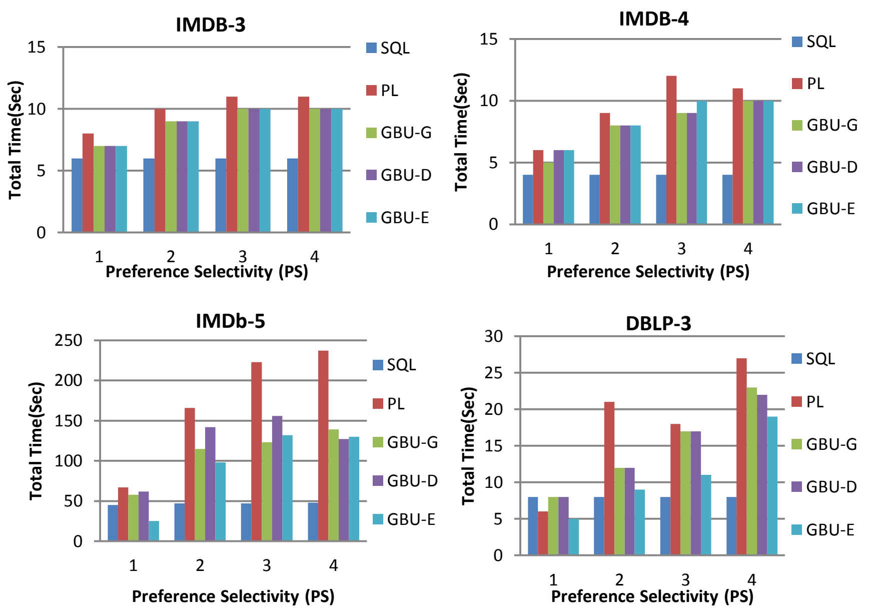

Fig. 10. Total execution time compared with selectivity of preferences.

Performance in relation to the distribution of preferences:

The final experiment examined how the distribution of preferences in the tables in query affects the cost of treating the query. Essentially, we modify the questions so that the preferences apply to one, several or all the tables of a query. Figure 11 shows the measured processing times by varying the number of tables related preferences. It appears that the processing times increase with the number of tables that are affected by preferences because of the cost valuation of preferences. Generally, variants of the algorithm GBU is among the most efficient test methods.

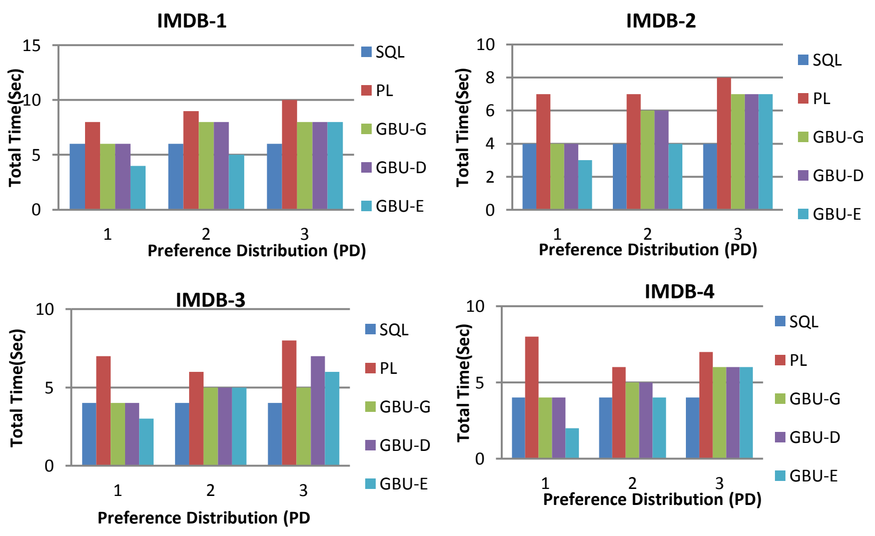

Fig. 11. Total execution time in relation to the distribution of preferences.

5. Conclusion

Evaluating the proposed cost model and optimization techniques, our experiments show that both the greedy algorithm and a dynamic programming algorithm generated execution plans perform quite well compared to the optimal plan. As expected queries having relatively small number of preferences or joins, dynamic programming algorithm performs slightly better than the greedy algorithm (the average percentage improvement measured was 4.32%). Conversely, for queries with multiple joins or a large number of preferences (e.g., *IP* |> 6) the greedy algorithm is usually preferred because of the lower cost optimization requires in relation to the dynamic programming algorithm.

References

1. Fürnkranz, J.; and Hüllermeier, E. (2010). Preference learning. *Encyclopedia of Machine Learning*. Springer US, 789-795.
2. Joachims, T.; (2002). Optimizing search engines using click through data. *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining.* Edmonton, AB, Canada, 133-142.
3. Harbaoui, A.; Sidhom, S.; Ghenima, M.; and Ghezala, H. B. (2014). Personalized Information Retrieval: Application to Virtual Communities, *Human Interface and the Management of Information*, 8521, 431-438.
4. Hristidis, V.; Koudas, N.; and Papakonstantinou. Y. (2001). Prefer: A system for the efficient execution of multi-parametric ranked queries. *In Proceedings of the 2001 ACM-SIGMOD Conference on the Management of Data*. Santa Barbara, CA, USA, 259-270.
5. Koutrika, G.; and Ioannidis, Y.E. (2005). Personalized queries under a generalized *preference* model, *Proceedings of the 21st International Conference on Data Engineering (ICDE 2005). Tokyo, Japan,* 841-852.
6. Holland, S.; Ester, M.; and Kieling. W. (2003). Preference mining: A novel approach on mining user preferences for personalized applications. *7th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD).* Dubrovnik, Croatia, 204-216.
7. Choi, W.; Liu, L.; and Boseon Yu. (2012). Multi-criteria decision making with skyline computation. *IEEE 13th International Conference on Information Reuse and Integration (IEEE IRI 2012),* Las Vegas, USA, 316-323.
8. Chengkai Li; Chang, K.C.; Ilyas, I.F.; and Song, S. (2005). RankSQL: Query algebra and optimization for relational top-k queries. *In Proceedings of the 2005 ACM-SIGMOD Conference on the Management of Data*. Maryland, USA. 131-142.
9. Sarawagi, S. (2008). Information extraction. *Foundations and Trends in Databases Archive*, 1(3), 261-377.
10. Ilyas, I.F.; Beskales, G.; and Soliman. M.A. (2008). A survey of top-k query processing techniques in relational database systems. *ACM Computing Surveys, (CSUR),* 40(4), New York, NY, USA.
11. Stefanidis, K.; Pitoura, E.; and Vassiliadis P. (2007). Adding context to preferences. *Proceedings of IEEE 23rd International Conference on Data Enginering (ICDE 2007),* Istanbul, Turkey, 846-855.

12. Adomavicius, G.; and Tuzhilin, A. (2005). Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *Proceedings of the 22nd International Conference on Data Engineering (ICDE 2006)*, Atlanta, GA, USA, 734-749.
13. Stefanidis, K.; Pitoura, E.; and Vassiliadis, P. (2007). Adding context to preferences, *IEEE 23rd International Conference on Data Engineering (ICDE 2007),* i*stanbul, Turkey*, 846–855.
14. Agrawal, R.; Rantzau, R.; and Terzi. E. (2006). Context-sensitive ranking. *In Proceedings of the 2006 ACM-SIGMOD Conference on the Management of Data,* Chicago, Illinois, USA, 383–394.
15. Werner K. (2002). Foundations of preferences in database systems, *Proceedings of the 28th international conference on Very Large Data Bases (VLDB 2002),* Hong Kong, China, 311–322.
16. Jan Chomicki. (2003). Preference formulas in relational queries, *Journal ACM Transactions on Database Systems (TODS 2003),* 4, 427–466.
17. Levandoski, J.; Mokbel, M.; and Khalefa. M.; (2010). FlexPref: A framework for extensible preference evaluation in database systems, *26th IEEE International Conference on Data Engineering. (ICDE 2010).* Long Beach, California, USA, 828-839.
18. Koutrika, G.; and Ioannidis. Y.E. (2004). Personalization of queries in database systems. *Proceedings of the 20th International Conference on Data Engineering.* Boston, MA, USA, 597-608.