A user-friendly framework for database preferences

Roxana Gheorghiu
University of Pittsburgh
Email: roxana@cs.pitt.edu

Alexandros Labrinidis
University of Pittsburgh
Email: labrinid@cs.pitt.edu

Panos K. Chrysanthis
University of Pittsburgh
Email: panos@cs.pitt.edu

***Abstract*—Data drives all aspects of our society, from everyday life, to business, to medicine, and science. It is well known that query personalization can be an effective technique in dealing with the data scalability challenge, primarily from the human point of view. In order to customize their query results, users need to express their preferences in a simple and user-friendly manner. There are two types of preferences: qualitative and quantitative. Each preference type has advantages and disadvantages with respect to expressiveness. In this paper, we present a graph-based theoretical framework and a prototype system that unify qualitative and quantitative preferences, while eliminating their disadvantages. Our integrated system allows for (1) the specification of database preferences and creation of user preference profiles in a user-friendly manner and (2) the manipulation of preferences of individuals or groups of users. A key feature of our hybrid model is the ability to convert qualitative preferences into quantitative preferences using intensity values and without losing the qualitative information. This feature allows us to create a total order over the tuples in the database, matching both qualitative and quantitative preferences, hence significantly increasing the number of tuples covered by the user preferences. We confirmed this experimentally by comparing our preference selection algorithm with Fagin's TA algorithm.**

I. Introduction

The supply and demand of data is becoming a commonplace in all aspects of our society; from everyday life (e.g., picking movies or restaurants), to business products, to medicine, and science in general. The term *"Big Data"* has been used to describe the challenges and opportunities from such a ubiquity of data, while also considering its volume, velocity, and variety characteristics. Although some may argue that Big Data is currently entering the *Trough of Disillusionment*, after following the typical Hype Cycle [1], the reality of the matter remains that there are still many technical challenges, as more and more people are accustomed to using data to drive their decisions and collaborations. Scalability is one such major challenge.

We distinguish two types of scalability:

- scalability from a *systems point of view* – this refers to traditional challenges due to the volume of data (and the rate of increase) and limitations in network bandwidth, processing, memory, and storage capacity. For example, how to make a single user query return all the results as fast as possible.
- scalability from a *human point of view* – given the volumes of data, new paradigms to aid in search are needed so that users do not get lost in a sea of data. For example, how to make a single user query return only the most relevant results for that user.

It is well-known that *query personalization* can be an effective technique in dealing with the scalability challenge, primarily from the human point of view. It is also well-known that when individual users within a group have the ability to personalized their work (i.e., see the content relevant to them) and shared their experiences, the collaboration within the group is significantly improved (e.g.,[2]). In order to personalize their query results, users need to provide their *preferences* in an effective manner (essentially letting the system form *user profiles*). These preferences are then used when users submit queries in order to only return the results that are most relevant to them. Cutting down the result set in this way improves both types of scalability.

There are two main types of user preferences defined in the literature [10]: *quantitative* and *quantitative*. *Quantitative preferences* are described by scores attached to each tuple that matches a preference. For example, consider the following preference: "I like comedies very much". This can be translated in the following quantitative preference: ("I like comedies", score =1) The score denotes users' interest in one or multiple data tuples. Using these scores we can define a total order over the database tuples, e.g., from the most preferred to the least preferred. *Qualitative preferences* are expressed as pairs of tuples. As an example, consider the preference "I like comedies more than dramas". This can be translated into the following qualitative preference: ("comedies", *preferred_over*, "dramas"). When put together, these pairs generally create only a partial order over database tuples, since some are incomparable.

Each type of preferences – quantitative and qualitative – has its advantages over the other. There are examples when a user's preference can be conveniently expressed using one approach but not the other. For example, it is very easy to express a negative preference in the quantitative model, by assigning a negative weight to that particular tuple or to a set of tuples that match a given predicate. However, there is no easy way to express a negative preference in the qualitative model, since this will require, for example, to explicitly list all tuples that are preferred over the non-preferred ones. In fact, this would need to happen for all tuples currently in the database and also for all tuples added later. Table I summarizes the key positive and negative aspects of the two different type of preferences.

Another important aspect of preferences comes from the

COLLABORATECOM 2014, October 22-25, Miami, United States

DOI 10.4108/icst.collaboratecom.2014.257659

TABLE I
QUALITATIVE, QUANTITATIVE AND THE HYBRID MODEL

Dimension	Qualitative Model	Quantitative Model	Hybrid Model
Generality	Mostly general	Less general; cannot express a preference like: "*From two movies with the same genre, I prefer the longer one*"	As general as possible
Intuitiveness	Easy and intuitive way to express preferences	Not easy to decide how to assign values	Any form can be used
Negative preference support	Not intuitively	Using negative values	Using negative values
Order created by the use of preferences	Partial	Total	Total
Combining preferences	By the use of different rules	Easily accessible using aggregation functions	Can use both aggregation functions and rules

fact that they can be expressed with different intensity levels. Preferences should not be seen as a binary option; instead, a system should allow every user to express his/hers preferences along with the intensity of that particular preference, i.e., how "strongly" he/she feels about that preference. This intensity value can be seen as a score attached to each tuple and it can easily be applied to a quantitative preference. But in the case of a qualitative preference, the intensity suggests the strength between two different tuples and cannot be associated, individually, to any of the two tuples involved in the preference; it should instead be linked with the pair of tuples.

The hypothesis of this work is that a hybrid model, which integrates qualitative and quantitative preferences by means of preference strength or intensity and user profiles, is both user-friendly and creates a global view of preferences that can be effectively used to rank the query results.

In this paper, we propose such a hybrid model which is able to overcompensate the negative aspects of one preference model by using the solutions provided by the other preference model. The formal underpinning of our proposed hybrid model is a *preference graph*. Each node in the graph represents a query predicate. We express quantitative preferences using edges that have the same starting and ending point. Qualitative preferences are represented by edges between two different nodes. Each edge is labeled with a value that represents the intensity of the preference. Users submit both qualitative and quantitative preferences along with an intensity value. In this way, individual users and groups create their own profile by incrementally adding or removing preferences over the database tuples. When a query is submitted, the system effectively selects the best combination of preferences from the user/group's profile to filter and rank the query results.

A. Contributions

In this paper we present our hybrid preference model and a prototype system that combine *quantitative* and *qualitative preferences* into a unified model using an acyclic graph, called HYPRE ([ˈhaipə]) Graph.

Specifically, this paper's contributions are as follows:

- We formally define our hybrid preference model and we show, using examples, how different types of preferences can be supported (Sec. II)
- We design and implement a prototype of a real system for our hybrid model and design algorithms to create and update the unified preference graph, while also detect and mark the conflicts (Sec. III).
- We show that our hybrid model can successfully map qualitative preferences into quantitative ones, using intensity values, hence allowing for significantly better "coverage" (up to 336%) of the database tuples (Sec. IV, Sec. VI).
- We experimentally show that our Practical and Efficient Preference Selection (PEPS) algorithm returns Top-K results correctly, while it also covers more tuples in the database that cannot be "seen" by Fagin's TA algorithm (Sec. III-E, Sec. VI).
- We showed that the model can be easily applied into a collaborative environment where the group preferences are used to supplement the lack of preferences for one particular user (Sec. III-F, Sec. VI).

II. UNIFIED PREFERENCE MODEL

A graph representation is the most natural formal way of exemplifying the connections between tuples in a database and visually depicting their relationships. By adopting the graph formalism, we can capture the two different preference approaches into our proposed unified model as well as record the strength/intensity of each preference.

Definition 1: Preference intensity is a decimal value between -1 and 1 and is used to express a negative preference, a positive preference or equality/indifference:

- Negative preferences are expressed using any value in [-1, 0); -1 is used to express complete dislike.
- Positive preferences are expressed using any value in (0, 1]; 1 is used to capture the most preferred tuple(s).
- Indifference : when used for quantitative preferences, 0 captures indifference towards a set of tuples.
- Equality : when used for qualitative preferences, 0 expresses preference equality between two sets of tuples.

TABLE II
THE DBLP RELATION

pid	Title	Year	Venue
t1	Personal Ontologies for Web Navigation.	2000	CIKM
t2	Composite Subset Measures	2006	VLDB
t3	Processing Proximity Relations in Road Networks	2010	SIGMOD
t4	Relational Joins on Graphics Processors	2008	SIGMOD

For a quantitative preference, the intensity value expresses the preference strength for one particular tuple (or set of tuples) over *all* other tuples in the database. In this case, intensity has the semantics of the score, and a large intensity value describes a strong preference towards that particular tuple (or set of tuples).

For a qualitative preference, the intensity value expresses the preference strength for one tuple (or set of tuples) over another tuple (or set of tuples). In this case, a small positive value will express a similarity on preferences (i.e., one tuple is almost as preferred as the other tuple).

Moreover, intensity can be seen as a constant value or as a function to allow dynamic ranking of preferences. As an example, consider the preference: "I like recent comedies", where recent can be expressed as a function on the year a movie was released and normalized in the proper range (i.e., [-1, 1]).

Definition 2: Hybrid Preference Graph HYPRE =(PV,PE) is a labeled directed and acyclic graph where:

- PV is a set of vertices where each vertex represents a tuple in the database or a query predicate.
- PE is a set of edges where each edge (v_i,v_j,s), $v_i, v_j \in$PV, defines a direction (i.e., from vertex v_i to vertex v_j), and is labeled with a score s. An edge from v_i to v_j captures a qualitative preference (i.e., the tuple(s) in vertex v_i is preferred over the tuple(s) in vertex v_j) whereas an edge from v_i to itself will describe a quantitative preference.
- The score s captures the preference intensity and is a value between -1 and 1, for the quantitative preferences, and between 0 and 1 for the qualitative preferences.

Our unified model uses a HYPRE graph to record users' preferences that can be viewed as triplets (v_i,v_j,s). When i≠j, the triplet (v_i,v_j,s) represents a qualitative preference, and when i=j, it represents a quantitative preference.

Instead of a predicate, each node in our graph can store only one tuple, in which case a tuple-based preference graph is created. A predicate-based preference graph (i.e., each node stores a predicate) is a scalable version of the tuple-based one especially for predicates that apply to a large set of tuples. Moreover, because a predicate can match only one tuple, we consider the tuple-based preference graph a special case of the predicate-based preference graph.

A. Examples of HYPRE Graph

Let us now exemplify how different types of preferences can be supported by our model using an instance of the *DBLP* database (see Table II). We start with an empty HYPRE graph and we incrementally add new preferences in the graph.

We again split the preferences into two categories:

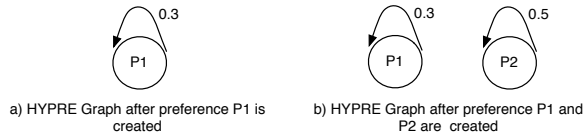
- quantitative preferences -any single node in the graph
- qualitative preferences -involving two different nodes

B. HYPRE Graph – Quantitative Preferences

Assume we have the following four preferences.

- P1: *"I prefer papers published between 2000 and 2005, with intensity 0.3"* -matching tuple(s): {t1}
- P2: *"I prefer papers published between 2005 and 2009, with intensity 0.5"* –matching tuple(s): {t2, t4}
- P3: *"I like papers published after 2009 with intensity 0.8"* -matching tuple(s): {t3}
- P4 (Negative Preference): *"I am not interested in papers published in INFOCOM."* -matching tuple(s): { }

Fig.1 shows the state of the graph after (a) preference P1 is inserted and (b) preference P2 is inserted in the graph.



a) HYPRE Graph after preference P1 is created

b) HYPRE Graph after preference P1 and P2 are created

Fig. 1. Quantitative Preferences

Preferences defined above (P1 to P4) are all quantitative preferences and one node is created in the HYPRE graph for each preference. Fig. 2 shows the entire graph, after all preferences P1 to P4 are inserted, where P1, P2, P3, P4 ∈ PV and e1=(P1, P1, 0.3), e2=(P2, P2, 0.5), e3=(P3, P3, 0.8), e4=(P4, P4, -1) ∈ PE.

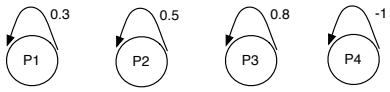


Fig. 2. All Quantitative Preferences

For the negative preference P4, there are no tuples to match it. However, if the database is updated, the HYPRE graph does not need any modification.

C. HYPRE Graph – Qualitative Preferences

For qualitative preferences we need to connect two different nodes. If the nodes are already part of the graph, we just add the directed edge to connect them. Else, if one or both nodes

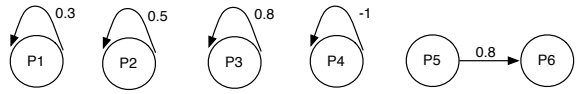


Fig. 3. Relative Preference (P5, P6)

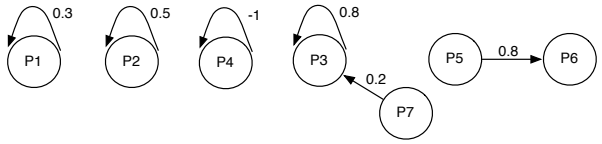


Fig. 4. Set Preference (P7,P3)

are not already part of the graph, we create the new nodes and connect them with a directed edge.

Qualitative preferences can be used to express multiple kinds of preferences as shown in the following examples:

- Relative Preferences: *"Given two VLDB papers, I prefer, with intensity 0.8, the one published in the last 4 years."* For this preference we create two different nodes: let P5 be the predicate:*[venue='VLDB' and year≥2010]* and P6 be the predicate:*[venue='VLDB' and year<2010]* where P5, P6 ∈ PV and e6 =(P5, P6, 0.8) ∈ PE. Fig. 3 is the new HYPRE graph after this qualitative preference is inserted.
- Preference Set: *"From a list of papers, I prefer VLDB papers and as many papers published after 2009."* Let P7 be the preference: *[venue='VLDB']* and since preference: *[year>2009]* already exists in the graph (i.e., node P3) we are going to use it to create this qualitative preference. Note that the user specifies that both preferences are important (paper should be published in VLDB and paper should be published after 2009) but since the year of publication is not as important (i.e., "as many as possible") we consider the intensity of this preference to be a small value (e.g., 0.2). Therefore we add one new node, P7 ∈ PV and e7 =(P7, P3, 0.2) ∈ PE, with the graph representation given in Fig. 4.
- Different Levels of Intensity: *"I really like papers published in SIGMOD but I prefer the papers published in VLDB a bit more than papers published in SIGMOD."* In the preference graph, we already have node P7, for the predicate: *[venue='VLDB']*. We are going to use this node for the left part of the qualitative preference and for the right side we create a new node, P8, with predicate: "I like SIGMOD papers, intensity 0.8". P7 and P8 ∈ PV and e8 =(P8, P8, 0.8), e9 =(P7, P8, 0.3) ∈ PE. As in the previous example, we map the expression "a bit more" to a small intensity value (e.g., 0.3).

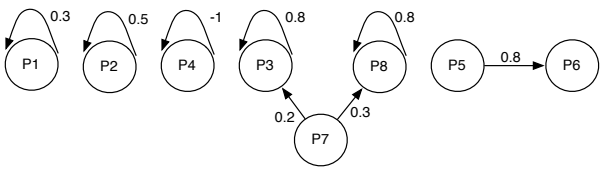


Fig. 5. HYPRE Graph with all preferences

III. HYPRE Graph Manipulation

A. HYPRE Graph Characteristics

Our solution for representing a HYPRE graph is based on a *graph database model* which is designed to provide efficient graph traversal and graph manipulation.

In this implementation, a node with no connections represents a quantitative preference and contains four properties: *(node_id, user_id, predicate, intensity)*, where *intensity* refers to the quantitative preference intensity. Two connected nodes creates a qualitative preference with the direction of the edge to define the prefer order between predicates and store the qualitative preference intensity. This implies that, if intensity values of these nodes exists, then the value of the node that has an outgoing edge (refer to as the *left node*) must always have a greater or equal intensity value with respect to the node where the edge ends (refer to as the *right node*).

Moreover, each edge has associated a label used to support graph traversals. The most common label is PREFERS, used to traverse the graph based on the partial order given by the qualitative preferences. Additionally, we use labels CYCLE and DISCARD to mark conflicts and inhibit traversal. We use CYCLE when a new inserted edge creates a cycle in the graph. We use DISCARD when a new edge causes the intensity value in the left node to become smaller than the intensity value in the right node and the system cannot recompute this value.

It should be noted that with this implementation, we can easily create only one graph for all users and groups and using the *user_id* property of a node, select all the nodes for a particular user/group, as needed.

B. Intensity Value Computation

To incorporate the nodes participating in a qualitative preference into the total order generated by the quantitative preferences we need to be able to convert the qualitative preferences into quantitative preferences. This process can be done if we can derive an intensity value for these nodes. However, in order for this to be correct, the new intensity value should be computed based on the existing qualitative preference intensity value and a quantitative preference intensity value (or a DEFAULT_VALUE if this does not exist).

For this purpose, we defined the following functions:

$$\begin{aligned} \text{IntensityL (left, ql, qt)} &= \min(1,\ qt*2^{\text{sign(qt)*ql}}) \\ \text{IntensityR (right, ql, qt)} &= \max(-1,\ qt*2^{-\text{sign(qt)*ql}}) \end{aligned} \quad (1)$$

where: *left/right* is the position of the node for which we computes intensity, *ql* is the intensity of the qualitative preference, and *qt* is the intensity of the quantitative preference.

Although these functions are not unique, they do need to have two important characteristics. First, if we compute the value of the left node, the returned value should be greater than the intensity value of the right node and it should directly depend on the intensity value of the qualitative preference. The stronger the qualitative intensity is – a stronger preference towards one set of tuples in comparison with a different set

of tuples – the greatest the returned value should be. On the other hand, if we compute the intensity value of the right node, we need to return a value that is smaller than the intensity value of the left node but it should also depend on the intensity value of the qualitative preference. Also, since the intensity value should always be in the interval [-1,1] we use -1/1 as the upper/lower-bound of IntensityL and IntensityR respectively. However, in practice the upper-bound should be a value smaller than the min/max value since these are very strong negative/positive opinions and as the value is computed by the system, we want to avoid artificially assigning the highest possible value.

C. Create HYPRE Graph Algorithm

The algorithm for creating the graph works incrementally. Given a list of preferences, both qualitative and quantitative, the algorithm inserts first all quantitative preferences, in a batch, and then all the qualitative preferences by searching if the nodes that define the qualitative preference already exist in the graph. Therefore, the algorithm necessitates two steps to insert all preferences.

Step 1. Algorithm creates a node for each quantitative preference defined. Assuming that predicates are unique, this will not create any conflict. In the case when the user provides a preference for the same predicate as one already inserted, the algorithm returns the *node_id* of the node that has the same *user_id* and *predicate* property values and updates the intensity value by computing the average of the two intensity values provided.

Step 2. Algorithm adds all qualitative preferences by inserting new nodes or using the ones already in the graph. For this case, there are three possible situations:

1. If the nodes are already in the graph, a directed edge is created between them and a conflict check routine is executed. A conflict appears when the intensity value of the left node is lower than the intensity value of the right node. In this case, the algorithm recomputes the intensity value for the node that is only connected to the graph through the new edge (i.e., has in_degree=0 and out_degree=1 OR in_degree=1 and out_degree=0). If both nodes are already connected to the graph, we keep the new edge only if it does not create a conflict. Otherwise we mark it as DISCARD. (To recompute the intensity value of a node that is already connected to the graph requires to check for possible conflicts generated by the new value. Although this is an interesting problem, it is beyond the scope of this paper and we will address it in our future implementation.)
2. If only one node is already in the graph the algorithm creates only one node, connects the two nodes with an edge, and recomputes the intensity value of the new node by using one of the functions defined in Eq. 1.
3. If none of the nodes that defines the qualitative preference are already part of the graph, the algorithm creates the two nodes, assigns a DEFAULT_VALUE to one of the nodes and computes the intensity value for the other node.

Moreover, if any edge introduced creates a cycle, the edge is kept in the graph, but is labeled with CYCLE to be avoided in the future graph traversals. However, if further updates on the graph eliminates this conflict, the edge label is switched back to PREFERS.

D. Preference Aware Query Enhancement

Once created, the preference graph is used to identify the relevant preferences and enhance the user-submitted query.

Different predicates can refer to the same or different attributes and can be combined using AND and OR semantics. There are three possible scenarios to combine SQL predicates. First we can add them all together as a disjunctive clause (i.e., connecting them with OR). In this case, the list of tuples returned will possibly be as large as the original dataset. Second, we can combine all the predicates in a conjunctive clause (i.e., combining them with AND) in which case, because many predicates can possibly refer to the same attribute, the returned list will become empty after relatively few steps. Finally, the third way is to combine predicates with the same attribute with an OR semantic and predicates with different attributes with an AND semantic. In this way, we can eliminate the empty result and the flooding problem associated with the previous two versions.

Stefanidis et. al [10] describe three different ways to compute the final intensity value when two or more quantitative preferences are combined: the *inflationary* strategy - the final score increases, the *reserved* strategy - the final score lies between the two values, and the *dominant* strategy - the highest value is used. For our model, we adopted the inflationary and reserved functions from Koutrika and Ioannidis' work [8]: $f_\wedge$ to calculate the combined intensity for conjunctive predicate combinations and $f_\vee$ to compute the combined intensity for disjunctive predicate combinations with the form given in Eq. 2 and Eq. 3. $f_\wedge$ behaves inflationary whereas $f_\vee$ has a reserved behavior.

$$f_\wedge(p_1, p_2) = 1 - (1 - p_1)(1 - p_2) \tag{2}$$

$$f_\vee(p_1, p_2) = \frac{p_1 + p_2}{2} \tag{3}$$

When we combine predicates with OR, the query returns tuples that match possibly, only the preference with the smaller intensity value. Since we do not know which predicate is matched, we penalize the final intensity value by assigning the average of the two. But, when we combine predicates with AND, the tuples in the result list are guaranteed to match all predicates. Because of that, the combined intensity is larger than the two given intensity values.

E. The PEPS Algorithm

The Practical and Efficient Preference Selection (PEPS) algorithm is our Top-K algorithm that returns the first k tuples selected by the best combinations of preferences in terms of combined intensity value.

The PEPS algorithm uses AND semantics to combine as many predicates as possible. To efficiently create valid combinations of preferences (i.e., combinations of predicates that

will return tuples) we have implemented an optimized version of the PEPS algorithm that makes use of a precomputed list of combinations of two predicates, that is updated when the preference graph is updated. Each item in the list contains the pair of predicates that are AND combined, the precomputed combined intensity value, and a count of number of tuples returned when the predicate combination is used. The PEPS algorithm uses this list to retrieve all the valid combinations that start with a particular predicate.

The algorithm iterates over the list of preferences and for a given preference p, it selects all the items from the list of combinations of two predicates, with the combined intensity value greater than the intensity value of p. This list is the starting point of the PEPS algorithm to expand them into multi-predicate AND-combinations. If the generated multi-predicate combinations do not retrieve all k tuples, the PEPS algorithm is invoked again with the next available preference, until all k tuples are retrieved.

PEPS uses two stacks. First stack, *CombStack* is initialize with all the combinations of two predicates found in a previous step. Second stack, *PrefStack*, is initially empty and it will be used to store all partial predicate combinations.

The algorithm starts by extracting one item from *CombStack* (e.g., a pair (p_i, p_j) where p_i and p_j each represents a different predicate). Then, if the *PrefStack* is empty, the algorithm creates a new partial combination by joining the two predicates with AND, computes a combined intensity value, and adds on the *CombStack* all valid combinations (p_j, X), where X is any predicate in the user's profile. When the *PrefStack* is not empty, the algorithm extracts the top predicate combination, appends the p_j predicate using AND and checks the validity of the new combination by verifying that there is a valid combination between all predicates already used and p_j. If the new combination is valid, then this combination is added to the *PrefStack* and all valid combinations of two predicates that start with p_j are also added to the *CombStack*.

If the new combination does not return any tuple, and the top of the *CombStack* does not contain a pair (p_i, Y), for any preference Y in the user profile, then the algorithm removes the last preference from the *PrefStack* and saves it to be used in subsequent invocations, if necessary. However, if the *CombStack* does contain a valid pair (p_i, Y), then this pair will be used to create a partial combination and its validity is checked again, as explained before. The algorithm continues until there are no more combinations left in *CombStack*.

F. The Collaborative PEPS Algorithm

In a collaborative environment users can be grouped together in order to access the knowledge (or the preferences) stored as a collection. Users can be grouped based on their common preferences or on their common interests. In the first case, for example, the users that have in common at least a predefined number of preferences can be grouped together. In the second case, the users' interests can be inferred in different ways. For example, the users that have a predefined number of common preferences with the same intensity value associated can be considered as having similar interests. Users can still be considered similar even when they do not have the required number of common preferences, if the results of a specific set of queries are similar when applying their preferences. Although the general problem of clustering or creating user groups based on their interest/preferences is orthogonal to this work, this motivated us to develop a collaborative version of PEPS to enhance the experience of users with limited number of preferences or with preferences that are too restrictive.

Our PEPS algorithm works the same whether the list of preference belongs to a single user or collected from multiple users who were assigned to the same group. It takes as input a list of preferences and decides which preferences to apply and prunes the ones that will not return a high enough combined intensity value. So the basic idea of the *Collaborative PEPS* algorithm is to select more preferences, other than the given ones of a user, allowing the system to better filter the result list for the user. In other words, given a user, Collaborative PEPS first creates a collaborative group for the user and then uses the group's list of preferences in returning the top k tuples.

Collaborative PEPS creates the user's group based on a slightly modified version of Jaccard similarity. The Jaccard similarity metric is computed using the ratio between the number of common items over the total number of items. However, we want that the selected users that have the highest number of preferences in common with the given user. Therefore, we fix the number of common preferences to the highest possible and then we select the users that maximize the Jaccard metric, that is, have the smallest number of preferences.

IV. Experimental Workload

For our experiments, we used the DBLP Citation Network V4 dataset [11] that contains both the DBLP dataset (2011 version) and information about citations. Data is organized in blocks, one block for each paper, which contains the title, author(s), venue, abstract, citations. By parsing this dataset, we create a database with four tables: *author(aid, full_name)*, *citation(pid, cid)*, *dblp(pid, title, venue, year, abstract)* and *dblp_author(pid, aid)*, with the cardinality and arity of each relation presented in Table III .

In addition to the relations given by the DBLP dataset, we create two more relations for to store preferences:

- *quantitative_pref(pfid, uid, predicate, intensity)*.
- *qualitative_pref(pfid, uid, leftPred, rightPred, intensity)*.

In both tables, the attributes *predicate, leftPred* and *rightPred* represent the SQL predicate that defines the preference, *uid* represents the author id and *intensity* is the strength of the preference.

A. Preference Extraction

To cover all possible types of preferences described in Sec. II-B and Sec. II-C, we designed the following preference extraction queries:

- Venue Preference (quantitative preference): Preference on the *venue* based on the venues where she published

TABLE III
STATISTICS FOR THE DBLP DATABASE

Relation	#Attr	Cardinality	
author	2	1,033,111	authors
dblp	5	1,614,306	papers
dblp_author	2	4,265,164	entries
citation	2	2,327,450	total entries
		316,562	distinct papers
quantitative_pref	4	10,361,592	entries
		1,033,010	distinct users
qualitative_pref	5	7,901,874	entries
		462,843	distinct users

- Author Preference (quantitative preference): Preference for an *author* based on the co-author information
- Preference of one author over another (qualitative preference): Author A is preferred over author B.
- Preference of one venue over another (qualitative preference): Venue X is preferred over venue Y.
- Negative Venue Preference (quantitative preference): For each user A, a negative preference towards the venues where she did not publish but other authors that were cited by A did publish.

We describe the details of preference extraction process and the formulas used to compute intensity values next.

1) Quantitative Preferences: Venue preference. The intensity is computed by, first, computing the total number of papers published by an author in one particular venue; then selecting the Top-5 most preferred venues and, finally, dividing the number of papers per venue to the total number of papers published in any of the Top-5 venues. We retained only the Top-5 results because the dataset contains, for each author, many singular papers per conference and hence, the intensity value becomes very small, close to zero, for most of the entries. As a reminder, a quantitative preference with intensity value equal to zero expresses user's indifference towards that particular set of tuples. Thus, the system can not benefit from having quantitative preferences with intensity equal to zero.

Author preference. Given the *Citation* relation, we find all authors that are cited by a given author. For each *uid*, we add one predicate for each author cited. The intensity value for each preference is computed by dividing the total number of citations of that particular author over the total number of papers cited.

Negative venue preference. For each user, we insert a negative preference towards a *venue* if the user never published in that particular venue but he cited authors that did publish in it. Given two authors, A and B, where author A cites author B, we extract a negative preference for author A, towards a venue where B published but A did not. The intensity value is computed as: (-1)***intensity**$_A(B)$***intensity**$_B(Venue)$,
where: **intensity**$_A(B)$ is A's preference intensity for author B and **intensity**$_B(Venue)$ is B's preference intensity for a particular venue. The reasoning behind this formula comes from the fact that if author A cited author B many times, and author B published in a venue multiple times (i.e., the intensity for that venue is close to 1) but author A never published in that venue, then author A should have a strong negative preference towards that venue. On the other hand, if author B published a lot in a venue where author A did not published, but author A is almost indifferent towards author B(i.e., intensity value is close to 0) then the preference's intensity should be a small negative value (close to 0).

TABLE IV
LIST OF PREFERENCES FOR USER_ID=2

Node_id	Preference	Intensity
7	dblp.venue="INFOCOM"	0.23
10	dblp.venue="PODS"	0.14
10372710	dblp_author.aid=128	0.19
10372711	dblp_author.aid=116	0.14

2) Qualitative Preferences: We create qualitative preferences over the authors (e.g., author A is preferred over author B) or over the venues (e.g., venue X is preferred over venue Y) using the already defined quantitative preferences over the authors and venues. The intensity value assigned is the difference between the intensity values of the two participating preferences. In some cases, the resulted intensity is a negative value which is taken into account when we insert the preference in the preference graph. Since our graph does not contain any negative qualitative preferences, we reverse the order of the preferences and use the positive value instead.

B. Preference Enhanced SQL Queries

As mentioned in the previous section, in order to avoid an empty result in the case of ad-hoc queries, we can combine preference predicates that are referring to the same attribute with OR semantics and preference predicates that are defined for different attributes with AND semantics.

For example, assume that a user with *user_id*=2 has submitted the following query: *"Show me all papers from the DBLP database."* Table IV displays a snapshot of one user's profile. Using the above OR-AND semantics rule, the final query enhanced with the user preferences, has the following form:

```
SELECT *
FROM dblp, author
WHERE (dblp.venue='INFOCOM' OR
                    dblp.venue='PODS')
  AND (author.aid=128 OR author.aid=116);
```

C. Collaborative Group Definition

In order to get a better insight of our proposed Collaborative PEPS algorithm, in our experiments we use two methods to create the collaborative group of a user. In addition to the Jaccard similarity method based on which we defined the Collaborative PEPS algorithm (III-F), we also use a simple method based on the common preferences. In this second method, we select a user with only one preferences and from the list of remaining users that have that preference in

TABLE V
INSERTION TIME

Insertion Type	Time (sec)	No. of pref
Quantitative Preferences	256.61	10,361,592
Qualitative Preferences	3680.26	7,901,874

common, we select the user for which the intensity value for the common preference is the same and has more than one preference.

V. EXPERIMENTAL TESTBED

We implemented our preference graph in a real system, using real data in order to evaluate its practicality and usefulness under realistic conditions. We store the preference graph using the Neo4j 2.0 engine and we use Java 1.7 to query both the graph database and the MySql database. We test our system using the data extracted from an extended version of the DBLP dataset [11], as discussed above.

In the previous section we described how we extracted preferences from the dataset. The extraction process was entirely done in MySql, and we used Neo4j, to create the nodes in our preference graph. For the quantitative preferences we read preferences from the relational database and create nodes in the preference graph in batch, reading 100,000 preferences at a time. Since we know that quantitative preferences are uniquely defined for each user, being able to create a batch insert into the preference graph speeds up significantly the time needed to create the graph. Table V shows the time necessary to create the preference graph for all users.

For the qualitative preferences however, since the preference nodes might have been already created, we had to use an individual approach - each qualitative preference is read from the relational database, the preference graph is searched for the existence of these preferences and new nodes are created, in case preferences do not already exist, or an edge is created between previously existing nodes, otherwise.

The DEFAULT_VALUE is used to generate the missing intensities, for all the qualitative preferences, and can be seen as a *seed* of the entire process. We only use this DEFAUL_VALUE if no other value is provided. In the process of generating the preference graph we experimented with different values for this *seed* -a default value of 0.5, the minimum value or the maximum value from all intensity values for one particular user, the minimum and maximum positive values. Among all this choices, we picked the average positive value as the starting point because it represents better the intensity values provided by one user. The DEFAULT_VALUE is computed for each user, therefore we are treating all users equally and there will be no user for whom the DEFAULT_VALUE will be outside of the range of values that she already provided.

VI. EXPERIMENTS

We designed our experiments to show first, the benefits of having both qualitative and quantitative preferences in a unified model and second, the key role of the intensity value.

A. Evaluation Metrics

In our experimental evaluation, we used three metrics: *coverage*, *similarity* and *overlap*.

Definition 3: Coverage The total possible number of tuples touched when all preferences are used independently.

Definition 4: Similarity Given two lists of tuples, the similarity metric returns the percentage of tuples that are common in the two lists.

Definition 5: Overlap Given two lists with the same tuples, L1 and L2, the overlap metric returns the percentage of tuples that are in the same relative order in both lists.

B. Benefits of a Unified Model of Preferences

Our model is using intensity values to combine the two preference types which, in the end, generates significantly more quantitative preferences, as presented in Fig. 6. For one particular user (uid=2, chosen at random), the graph shows that initially there are 36 quantitative preferences, but after inserting all qualitative ones, the preference graph will contain 172 nodes.

By using the formulas presented in Sec. III, we are able to assign a intensity value to all nodes connected in the graph by generating a carefully selected default value follow by computing a value given the intensity value of the qualitative preference and an existing intensity value for one of the predicates (i.e., the left or the right node that creates the qualitative preference). This way we can transform all qualitative preferences in quantitative preferences, without loosing the underline information provided by a qualitative preference, that will still be stored in the graph.

With more quantitative preferences we can overall cover more tuples in the database. Fig. 7 shows the number of distinct tuples returned if we run:

1. Only original preferences: (QT) only quantitative preferences, (QL) only qualitative preferences, (QT+QL) both qualitative and quantitative preferences.
2. All preferences extracted from our HYPRE Graph.

For the first case, qualitative preferences in the original form have only information about the intensity of the preference, about how much one set of tuples is preferred over the other. Because of that, if the intensity provided was strictly greater than zero we ran only the left preference since we only know that left is preferred over right. However, when intensity is equal to zero we ran both left and right preferences since zero intensity, in the case of a qualitative preference, means that both set of tuples are equally preferred. Fig. 7 shows that, for both users, our model can cover significantly more tuples from the database due to our mechanism that transforms a qualitative preference into two different quantitative preference. This improvement is from 120% compared to both quantitative and qualitative (uid=388437) up to 336% compared to just quantitative preferences (uid=2). Of course, more results in this case means better results because we are able to order them according to the users' preferences.

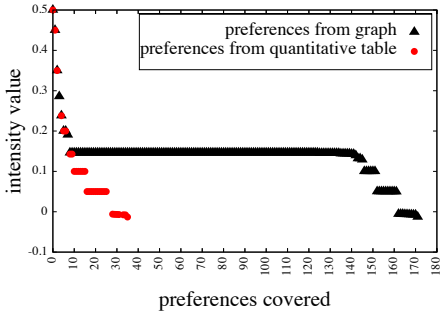


Fig. 6. QT for uid=2

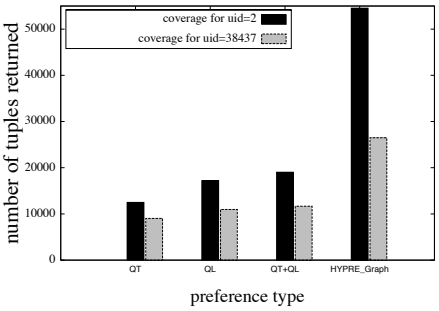


Fig. 7. Coverage for uid=2 and uid=38437

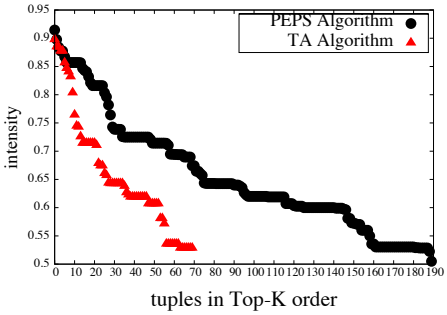


Fig. 8. Coverage -PEPS vs. Fagin's TA algorithm

C. Evaluation of the PEPS Algorithm

In order to evaluate our Top-K (PEPS) algorithm's correctness, we implemented the well known TA algorithm ([3]) by generating, for different users, the combined intensity value for each paper. TA algorithm assigns different scores to each tuple in the database based on the attributes used in preference's predicates. In our dataset, there are two types of predicates - on the *venue* and on the *author*. Because of that, we created two different tables *intensity_author* and *intensity_venue* with three attributes: (user_id, paper_id, combined_intensity). The combined intensity values, in both tables, were computed using Eq. 2. Finally, we combined the intensity values from the two tables to return a final score for each tuple. The final ranking given by the TA algorithm was used to evaluate the efficiency of our algorithm.

Because Top-K algorithms work only for the quantitative preferences, we first created a HYPRE graph that stores only the quantitative preferences. We ran PEPS over this graph and we compared our results against those of the TA algorithm. The results show 100% similarity and 100% overlap.

In order to assess the advantages of PEPS when qualitative preferences are considered, we ran PEPS over the large HYPRE Graph, containing both qualitative and quantitative preferences. This time, we looked at the ranking of tuples with combined intensity value at least as high as the maximum preference intensity value for user with uid=2 (i.e., 0.5). The results depicted in Fig. 8 show the two major advantages of our Top-K algorithm.

1. The PEPS algorithm offers better coverage, i.e., finds more tuples than the TA algorithm with intensity value higher or equal to 0.5.
2. Overall, the PEPS algorithm returns tuples with higher intensity value than the TA algorithm.

These advantages are a result of the fact that PEPS has access to more preferences than the TA algorithm and since these preferences are derived from both quantitative and qualitative preference have in general higher intensity values than the quantitative preferences used by the TA algorithm. Moreover, when looking at the similarity between the two returned Top-K lists, we found that there was 37% of matching tuples. To measure the overlap between the two lists, we first extracted the matching tuples from the two lists and then we verified that their order is preserved across the two lists. Again, there was a 100% match between the two list as in our first experiment.

These two experiments show that our solution is not only performing as good as the TA algorithm – we have a perfect match when only quantitative preferences are used – but it also performs better overall because it has the advantage of using the qualitative preferences too. Furthermore, it does not incur any performance penalty. For example, for Top-800, PEPS takes on average 2 sec to run for this workload. We measured the time complexity of our algorithm by varing the size of K, from 10 to 800, in 100 increment, and record the execution time. We repeated this process 10 times and we averaged the response time for each K value in order to eliminate any time variations due to I/O requests.

D. Evaluation of the Collaborative PEPS Algorithm

In order to exemplify how the system performs for groups, we selected a user with a low number of preferences (uid=236892, number of preferences=4). As discussed in Sec. IV-C, the selection of the user who participates in the group is based on a slightly modified version of Jaccard measure. The new user (uid=115539) selected to be in the group has the most number of preferences in common with the given user (i.e., three preferences) and the least number of preferences overall (43 preferences). This way we maximize the number of common preferences and the similarity between the two users based on Jaccard measure.

We ran PEPS with preferences from only the first user and then Collaborative PEPS with group preferences, i.e., PEPS with preferences from both users. We measured again the similarity and overlap between the returned list of tuples. The results showed that there is a 65% match in terms of similarity and 100% overlap. The group preferences brought overall more tuples in the result list (9702 compared to 6324). However, looking at the final intensity value assigned to each tuple, in the two lists, there is no difference which means either that the tuples returned by the common preferences are

not "touched" by other group preferences or the tuples returned have already the maximum possible intensity value.

For the second experiment of comparing the algorithms, we selected a user with only one preference (uid=158) and for the new user, that will participate in the group, we selected user with uid=117965 because it has the same intensity value for the common preference. For this two users, the results showed that there is only 15% match in terms of common tuples returned (857 for the first user versus 5568 using the group preferences) and 100% overlap. However, there are four tuples that received a higher final intensity value. Using only the one preference available, the tuples receive a combine intensity value of 0.33. In the second run, because these tuples matches more preferences, the final intensity value becomes 0.55 for two tuples and 0.70 for the other two.

VII. Related Work

In the database domain preferences are seen as *soft* criteria and are used to filter the data to avoid information overload. In contrast, predicates in the SQL WHERE clause are seen as *hard* constraints and a non-empty result is returned only when all conditions are met. Many solutions have been proposed for working with preferences; Stefanidis et al[10] is a comprehensive survey. In most cases the designed systems can handle only one type or preference (e.g., qualitative *or* quantitative). Our proposed model combines these two different approaches into a unified model, whose main theoretical idea was concurrently introduced in [5], [4]. In this paper, we present the complete framework (including the algorithms to compute intensity), a practical implementation, two new metrics: coverage and utility, and an experimental evaluation using a real data set.

Kiessling et al. [6] proposed a framework that can support a hybrid version of both qualitative and quantitative preferences. The PreferenceSQL [7] system introduces a new clause, PREFERRING, in which the user can state her preferences relative to the current query. All preferences are connected with an AND operator except for the case when a qualitative preference is defined, in which case a PRIOR TO operator is used. In this framework users need to fully describe their preferences for each query.

Our work differs from Kiessling et al. because each preference is enhanced with intensity information to allow an ordering over the database tuples in the query result. Our work also handles both qualitative and quantitative preferences through user profiles. Both dimensions are important when we consider preferences because, together, they can generate a global ranking of tuples by (1) adding intensity values to every tuple for which a preference can be applied that can be further used to rank the query results and (2) dynamically modify the intensity values of tuples when a new preference is introduced in the profile and that is connected to the already existing ones.

The work done by Koutrika and Ioannidis [9] is the other most related to ours. In their work the preferences are kept as query predicates with intensity values attached. In contrast to our work, they only record quantitative preferences and they are using them to create a preference network (i.e., a directed acyclic graph) that will allow an efficient identification of relevant preferences. This graph is used to depict the relation between preferences (i.e., each node in the network refers to a subclass of entities that its parent refers to) whereas in our case the graph's edges depict the flow of the preferences from the most preferred to the least preferred.

In contrast to [9], our work keeps track of preferences in any form (qualitative and quantitative) and our graph representation captures user specific order of tuples as they will show up in the final response after preferences are applied.

VIII. Conclusions

In this paper we presented a new framework (HYPRE Graph) that incorporates qualitative and quantitative preferences in a hybrid, unified model using intensity values to capture the strength of both preference types. We implemented our framework in a real graph database system (Neo4j) and experimentally evaluated it using real data extracted from DBLP. In terms of coverage, which captures how many tuples are "touched" by a user preference, our HYPRE graph model can outperform the other alternatives from 120% up to 336%. Our experimental results also showed that our Top-K algorithm (PEPS) that utilizes the HYPRE graph returns the same results as Fagin's algorithm, when using only quantitative preferences, but significantly outperforms it for the full graph. Finally, we proposed and experimented with a collaborative version of PEPS which dynamically enhances the preferences of a user with those of other similar users.

IX. Acknowledgments

We would like to thank Christos Faloutsos, Wolfgang Gatterbauer, and Andy Pavlo for the fruitful conversations and insightful feedback they gave us on this work. This work was funded in part by NSF awards OIA-1028162 and IIS-0746696.

References

[1] http://en.wikipedia.org/wiki/hype_cycle.

[2] C. E. Evangelou, M. Tzagarakis, N. Karousos, G. Gkotsis, and D. Nousia. Augmenting collaboration with personalization services. *IJWLTT*, 2(3):77–89, 2007.

[3] R. Fagin, A. Lotem, and M. Naor. Optimal aggregation algorithms for middleware. In *PODS*, pages 102–113, 2001.

[4] R. Gheorghiu. *Unifying Qualitative and Quantitative Database Preferences to Enhance Query Personalization*. PhD thesis, September 2014.

[5] R. Gheorghiu, A. Labrinidis, and P. K. Chrysanthis. Database preferences – A unified model. In *PersDB*, 2012.

[6] W. Kiessling. Foundations of preferences in database systems. In *VLDB*, pages 311–322, 2002.

[7] W. Kiessling and G. Köstler. Preference SQL: design, implementation, experiences. In *VLDB*, pages 990–1001, 2002.

[8] G. Koutrika and Y. Ioannidis. Personalization of queries in database systems. In *ICDE*, pages 597–608, 2004.

[9] G. Koutrika and Y. Ioannidis. Personalizing queries based on networks of composite preferences. *ACM TODS*, 35(2):13:1–13:50, 2010.

[10] K. Stefanidis, G. Koutrika, and E. Pitoura. A survey on representation, composition and application of preferences in database systems. *TODS*, 36(3):19:1–19:45, 2011.

[11] J. Tang, J. Zhang, R. Jin, Z. Yang, K. Cai, L. Zhang, and Z. Su. Topic level expertise search over heterogeneous networks. *Machine Learning Journal*, pages 211–237, 2011.