UNIFYING QUALITATIVE AND QUANTITATIVE DATABASE PREFERENCES TO ENHANCE QUERY PERSONALIZATION

by

Roxana Gheorghiu

B.Sc. in Computer Science, University of Bucharest, 2004

M.Sc. in Computer Science, University of Bucharest, 2005

M.Sc. in Computer Science, University of Pittsburgh, 2013

Submitted to the Graduate Faculty of

the Kenneth P. Dietrich School of Arts and Sciences in partial

fulfillment

of the requirements for the degree of

Doctor of Philosophy

University of Pittsburgh

2014

UNIVERSITY OF PITTSBURGH

KENNETH P. DIETRICH SCHOOL OF ARTS AND SCIENCES

This dissertation was presented

by

Roxana Gheorghiu

It was defended on

May 30, 2014

and approved by

Alexandros Labrinidis, Associate Professor, University of Pittsburgh

Panos K. Chrysanthis, Professor, University of Pittsburgh

Adam J. Lee, Computer Science Department

Vladimir I. Zadorozhny, School of Information Sciences

Dissertation Advisors: Alexandros Labrinidis, Associate Professor, University of Pittsburgh,

Panos K. Chrysanthis, Professor, University of Pittsburgh

UNIFYING QUALITATIVE AND QUANTITATIVE DATABASE PREFERENCES TO ENHANCE QUERY PERSONALIZATION

Roxana Gheorghiu, PhD

University of Pittsburgh, 2014

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 personalize their query results, user's need to express their preferences in an effective manner. There are two types of preferences: qualitative and quantitative. Each preference type has advantages and disadvantages with respect to expressiveness. The most important disadvantage of the quantitative model is that it cannot support all types of preferences while the qualitative model can only create a partial order over the data, which makes it impossible to rank all the results. The hypothesis of this dissertation is that it is possible to overcome the disadvantages of each preference type by combining both of them, in a single model, using the notion of intensity. This dissertation presents such a hybrid model and a practical system that has the ability to convert the intensity values of qualitative preferences into intensity values of quantitative preferences, without losing the qualitative information. The intensity values allow to create a total order over the tuples in the database that match a user's preferences as well as to significantly increase the coverage of preferences. Hence, the proposed model eliminates the disadvantages of the existing two types of preferences. This dissertation formalizes the hybrid model using a preference graph and proposes an algorithm for efficient preference combination, which is evaluated in an experimental prototype. The experiments show that: (1) intensity plays a crucial role in determining the order of selecting and applying the preferences, and simply ordering the preferences based on the intensity value is not necessarily sufficient; (2) the model can achieve three orders of magnitude increase in coverage compared to other alternatives; (3) the solution proposed outperforms other Top-*k* algorithms by being able to use both qualitative and quantitative preferences at the same time, and (4) the algorithm proposed is efficient in terms of time complexity, returning tuples ordered by the intensity value in a matter of seconds.

Keywords Qualitative Preferences, Quantitative Preferences, Top-K Ranking.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ALGORITHMS

LIST OF EQUATIONS

1.0 INTRODUCTION

1.1 MOTIVATION

The supply and demand of data is becoming commonplace in all aspects of our society; from everyday life (e.g., picking movies or restaurants), to business (e.g., advertising and marketing campaigns), to medicine (e.g., high-throughput sequencing), and science in general (i.e., The Fourth Paradigm [18]). The term *"Big Data"* [30] 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 in their lives (for personal, business, or scientific reasons), making scalability a major challenge.

In the database domain we distinguish two types of scalability related to the Big Data Paradigm:

- 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([29], [15], [33]) that *query personalization* can be an effective technique in dealing with the scalability challenge, primarily from the human point of view. In order to person-

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

alize 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, mentioned above.

There are two main types of user preferences defined in the literature [40]: *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.

While *qualitative* preferences are more user-friendly because it allow preferences described by comparison, *quantitative* preferences are easier to use by a system since a ranking of the tuples that mach any preferences is done automatically using the attached score.

Due to the importance of preferences, especially in the context of Big Data, a plethora of alternatives have been proposed. Table 1 contains a summary of the most important preference solutions and their main characteristics. The majority of the work done so far can handle only one preference type (mostly qualitative since is the most general approach), and at the tuple granularity level (i.e., preferences expressed on the value of the tuples's attribute) [40].

1.2 QUALITATIVE VS. QUANTITATIVE PREFERENCES

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

Table 1: Existing Preference Representations, as Presented in Pitoura et al. [40]

		Formulation		Granularity			Context		
		Qualitative	Quantitative	Tuple	Attribute	Relationship	Context-free	Internal	External
1	Lacroix and Lavency 1987 [27]	✓		✓			✓		
2	Agrawal and Wimmers 2000 [2]		✓	✓			✓		
3	Kießling 2002 [21]	✓	✓	✓			✓		
4	Chomicki 2002 [8]; 2003 [9]	✓		✓			✓	✓	
5	Holland and Kißling 2004 [19]	✓		✓					✓
6	Koutrika and Ioannidis 2004 [24]; 2005 [25]		✓	✓		✓	✓		
7	Agrawal et al. 2006 [1]	✓		✓				✓	
8	Endres and Kißling 2006 [13]	✓		✓				✓	
9	Bunningen et al. 2006 [46]		✓	✓					✓
10	Stefanidis et al. 2006 [41]; 2007 [42]		✓	✓			✓		✓
11	Mindolin and Chomicki 2007 [32]	✓		✓	✓			✓	
12	Ciaccia 2007 [10]	✓		✓				✓	
13	Georgiadis et al. 2008 [16]	✓		✓	✓		✓		
14	Miele et al. 2009 [31]		✓	✓	✓				✓

the other. For example, it is very easy to express a negative preference in the quantitative model, by assigning a negative score 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.

The basic idea of the use of preferences is to generate an order over the database tuples, ranking

Table 2: Qualitative, Quantitative and the Hybrid Model

Dimension	**Qualitative Model**	**Quantitative Model**	**Hybrid Model**
Generality	Mostly general	Less general	As general as possible
Intuitiveness	User friendly Easy and intuitive	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

them from the most preferred to the least preferred with respect to a user. Tuples that do not match any preference can be divided into two categories: equally preferred and incomparable. In the case of quantitative preferences, tuples that are equally preferred can be seen as having the same score, whereas, in the case or qualitative preferences, tuples that are incomparable cannot be included in the partial or total order defined by the preferences.

Another important aspect of preferences comes from the 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.

This intensity can also change given the query context. For example, during a rainy day someone will be very happy with a movie recommendation but during a sunny day, the same person may be more interested in outdoor activities and less interested in movies.

Table 2 summarizes the key the positive and negative features of the two type of preferences that motivated our work to seek a user-friendly way to express preferences which enables the creation of total order over the tuples.

The most important negative aspects of the *Qualitative Model* are:

- Inability to easily express a negative preference.
- It can only create a strict partial order between tuples (i.e., a partial order that is irreflexive).
- It can be seen as supporting only local preferences since a qualitative preference expresses the selection of one set of tuples, when compared with another set of tuples. A qualitative preference does not make any statement about a user's willingness of selecting one particular tuple out of *all* database tuples.

Although the *Quantitative Model* can be seen as supporting global preferences, it is less general than a qualitative model therefore it cannot express all preferences that a qualitative model can. For example, it cannot express the following preference: "Given two movies with the same genre, I prefer the longer one". Moreover, although this model is very intuitive and easy to use after scores have been attached to each tuple, there is no intuitive way to define how these scores should be assigned.

1.3 STATE OF THE ART AND ITS LIMITATIONS

Given the complementary advantages of the qualitative and quantitative models an attempt was made by Kiessling et al. [23], in Preference SQL (see Table 1) to combine qualitative and quantitative preferences. Preference SQL is a framework that can support a hybrid version of both qualitative and quantitative preferences. Each preference is submitted by the user at the query time in a special clause called *PREFERRING* and can be seen as a local view of preferences, since for each query submitted, the user needs to define their preferences. All preferences are connected with an AND operator except for the case when a qualitative preference is defined, in which case an ELSE operator is used (e.g., "PREFERRING venue IN ('CIKM') ELSE ('SIGMOD')"). Moreover, to set a priority between two different preferences, a new operator, PRIOR TO, is used (e.g., "PREFERRING venue IN ('CIKM') ELSE ('SIGMOD') PRIOR TO year $>$ 2010"). In this framework users need to fully describe their preferences for each query.

The Preference SQL system also allows users to define their own scoring function that will virtually assign a score to all tuples returned by the query. This feature gives users the chance to define their own quantitative preference and their own scale for preference intensity. However, this does not capture the strength of a *qualitative* preference and therefore it cannot be used to define a qualitative preference.

Preference SQL is the first system that combines the two different types of preferences but it does not completely solve the problem because:

(a) Preferences need to be manually introduced by each user every time a query is submitted. Although there are multiple features that can be used to create a query that will combine user's preferences, one can easily see that it can quickly become cumbersome, when many preferences are used. Not only that user needs to remember all the preferences, but also, enhancing a query with many preferences, every time the query is ran, is a task that is time consuming and prone to errors.

(b) In the case of a qualitative preference, there is no solution to capture the preference's strength. The Preference SQL system allows users to define which tuples are preferred but it cannot capture the strength of that preference. Qualitative preferences can define an equally preferred set of tuples or a ranking of one set over the other. For example,

P_1: "Color red is $\underline{\text{much more}}$ preferred than color blue". This is a preference that ranks tuples with color='red' over tuples with color='blue'. In Preference SQL this can be express as: "PREFERRING color in ('red') ELSE color in ('blue')".

P_2: "I like color blue and red equally". This is a preference that defines an equality in preference between tuples with color='blue' and color='red'. In Preference SQL this can be express as "PREFERRING color='red' AND color='blue' ".

P_3: "I like $\underline{\text{slightly better}}$ the color red then color blue". This is, again, a preference that ranks tuples with color='red' over tuples with color='blue' however, there is no difference in the way the clause is defined in Preference SQL, when compared with P_1. However, preference P_1 expresses a strong preference whereas this preference is a weaker preference and the tuples that match it can be almost seen as equally preferred. Because of the lack of intensity support, this information will be lost.

This lack of scoring of a binary relation when describing a qualitative preference, can lead to an unexpected ordering of query results.

1.4 THE PROPOSED HYBRID MODEL

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 dissertation we present a hybrid preference model and a prototype system that combine *quantitative* and *qualitative* preferences using their intensity values, providing a better approach then Preference SQL. The formal underpinning of our proposed unified 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, users 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 users profile to filter and rank the query results.

Although we define only predicate-based preferences, is interesting to see that attribute-based preferences can be used to support skyline queries by defining a preferences on the attribute with a function associated. For example, the following preference "I want the cheapest hotel that is close to the beach" can be express by using two attribute nodes (i.e., <distance, min> and <price, min>) where *min* is the function associated with the attribute. Also, we can create qualitative preferences by assigning an order over the attribute nodes, and decide which one is more important (e.g., price is more important than distance).

1.5 CONTRIBUTIONS

In this dissertation we create a new and holistic model to capture database preferences. Our model combines qualitative and quantitative preferences; stores, combines and assigns intensity values for each preference; provides a new algorithm that is guaranteed to return a list of combined preferences that can be further used to retrieve tuples in descending order of intensity.

In the database domain preferences are seen as *soft* criteria and are used to filter the data to avoid information the starvation problem (no tuples returned) or the flooding problem (too many tuples returned). 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.

In summary, the key contributions of this dissertation are:

1. A theoretical model, called HYPRE ([‘haipə]), that describes the preference graph and how different types of preferences can be specified.
2. A method to convert qualitative preferences to quantitative preferences which are incorporated into SQL statements as both hard and soft constraints and a Top-K algorithm which utilizes this method.
3. An experimental evaluation of the time complexity to create the preference graph and a theoretical proof of the complexity of preference combination problem.
4. Evaluation of our solution and validation of two different hypotheses. First, a unified model significantly increases the number of quantitative preferences available and it allows to totally order the tuples in the database based on the intensity of preferences. Second, the order in which the preferences are combined is very important.
5. Evaluation of our system in comparison with a well known Top-K algorithm that shows that our system consistently order the tuples as the Top-K algorithms, without a performance penalty. Moreover, we show that our model covers more tuples in the database by having access simultaneously to both qualitative and quantitative preferences.

1.6 OUTLINE

The rest of this dissertation is structured as follows: In Chapter 2 we provide the necessary background for Database Preferences and Graph Preferences and describe the characteristics of database preference systems that already exist in the literature. In Chapter 3 we describe our theoretical model for unifying qualitative and quantitative preferences followed by a discussion about how we support and implement this model in a real system, in Chapter 4. In Chapter 5 we introduce three approximation algorithms that are used to extract and combine preferences followed by two *practical and efficient* preference combination algorithms – the complete and the approximate versions. In Chapter 6 we discuss about the preference extraction process and the two metrics – *utility* and *coverage* – we defined to characterize the results and we present the findings of running different experiments on a real dataset in Chapter 7. Finally this dissertation is concluded in Chapter 8 by a guideline for future improvements and feature enhancements of the model and the system, as well as a summary of the overall work.

2.0 BACKGROUND AND RELATED WORK

In this section we first describe the two different types of database preferences known in the literature in terms of representation and composition, with the emphasis on their positive and negative aspects. Then we take a closer look at different models defined in the literature, showing how our work differs and why it is an improvement over the existing work.

In the following two sections we will make use of the following notations: R (A_1, A_2, …, A_n) is a relational schema with *n* attributes denoted by A_1, A_2, … A_n. For each attribute, we define the space of possible values of that particular attribute by *dom*(A_i). Moreover, a tuple *t*, in R, is defined as t=(a_1, a_2, …, a_n) where $a_i \in$ dom(A_i). Using these notations, we next define quantitative and qualitative preferences.

2.1 QUANTITATIVE PREFERENCES

Let us consider the relational schema and the tuples given in Table 3 with their associated intensity values given in Table 4.

Definition 1. *Quantitative preferences – Given a relational schema R and D, a subdomain of the real numbers set* $\mathbb{R}$*, a quantitative preference is defined as a function p:dom(*A_1*)* x *dom(*A_2*)* x … x *dom(*A_n*)* $\rightarrow$ *D. Given tuple t* $\in$ *R, p(t) represents the score (or the intensity) associated with the tuple t.*

Intuitively, a score[1] attached to one tuple describes a quantitative preference. As an example, consider the preference statement: "I like comedies very much" and D=[-1,1]. This can be trans-

[1]We use *score* and *intensity* interchangeable.

lated in the following quantitative preference: ("I like comedies", score =1). The score denotes users' interest in one or multiple data tuples that match a specified condition. Using these scores we can easily define a total order over the set of tuples that have a score attached. All other tuples cannot be considered for the total order until they also get a score (e.g., tuple m6 in Table 4).

Example 1. *Using the values provided in Table 4 we can see that tuple* m2 *is preferred over tuple* m5 *which, in turn, is preferred over tuples* m1 *and* m4.

In the example above, tuples *m4* and *m1* have the same intensity values whereas there is no intensity value defined for tuple *m6*. When we discuss about preferences, some tuples in the database will be equally preferred whereas for other tuples the user will have an indifferent attitude.

Definition 2. *Equally preferred – Given a relational schema R and D, a subdomain of the real numbers set $\mathbb{R}$, a quantitative preference* **p** *and two tuples, t_1 and $t_2 \in R$ (or two sets of tuples in R), we say that t_1 is equally preferred to t_2 if and only if* **p**(t_1)= **p**(t_2).

Example 2. *In Table 4, tuples m4 and m1 have the same intensity value therefore they are equally preferred.*

Definition 3. *Indifference – Given a quantitative preference* **p** *and a set of tuples T=$\{$ t, t $\in R\}$, we say that any* t $\in$ *T is in an indifference preference relation if and only if* **p**(t) *is equal to zero.*

Example 3. *In Table 4, tuple m3 has an intensity value equal to zero. This value is used to mark an indifference towards a particular tuple or set of tuples.*

2.2 QUALITATIVE PREFERENCES

Definition 4. *Qualitative preferences.*

Given a relational schema R, a tuple-level qualitative preference is defined as a binary relation $\succ_p$ over $\prod_{i=1}^{n} dom(A_i)$ x $\prod_{j=1}^{n} dom(A_j)$. Given tuples $t_1 \in \prod_{i=1}^{n} dom(A_i)$ and $t_2 \in \prod_{j=1}^{n} dom(A_j)$, $t_1 \neq t_2$, the pair (t_1, t_2) represents a single qualitative preference, $t_1 \succ_p t_2$, and is read as t_1 is preferred over t_2.

Intuitively, qualitative preferences are expressed as pairs of tuples where the first tuple in the

Table 3: The Movie Relation

movie_id	**title**	**year**	**director**	**genre**
m1	Casablanca	1942	M. Curtiz	drama
m2	Psycho	1960	A. Hitchock	horror
m3	Schindler's List	1993	S. Spielberg	drama
m4	White Christmas	1954	M. Curtiz	comedy
m5	The Adventures of Tintin	2011	S. Spielberg	comedy
m6	The Girl on the Train	2013	L. Brand	thriller

Table 4: Intensity

movie_id	**score**
m1	0.3
m2	0.9
m3	0
m4	0.3
m5	0.6
m6	

pair is the one preferred when compared to the second tuple in the pair. As an example, consider the preference "I like comedies more than dramas". This can be translated into the following qualitative preference: ("comedies", "dramas") or "comedies" $\succ$ "dramas". Following the relational schema in Table 3, the same preference can be stated as: $\{$m5, m4$\} \succ_p \{$m1, m3$\}$.

In the literature there is no clear definition for specifying an equality preference. In principle, two tuples are equally preferred if they are equivalent to each other and they can substitute each other in the list of results without changing the semantics. Usually tuples that are not compared to each other in any preference are considered to be equally preferred but there are cases when they are instead *incomparable* and they cannot, in some fundamental sense be compared to each other. Unless is otherwise stated, it is very hard to distinguish between these two cases and most of the time an a-priori assumption is made in terms of how these tuples will be treated.

Definition 5. *Indifference preference relation – Given a tuple* t *in R, we say that t is in an indifference preference relation with any other tuple in R if and only if* $\forall t_i \in R \nexists$ *preference* p *such that (t* $\succ_p t_j$ *or* $t_j \succ_p$ *t). In other words, tuple t is not part of any preference relation.*

Example 4. *A user defines the following preferences:* P_1*:*"Given two drama movies, I prefer the most recent one." , P_2*:* "I prefer a comedy movie directed by Curtiz to a comedy directed by Spielberg" *and* P_3*:* "If two movies are directed by Curtiz, I prefer the newer one". *Using the relational schema shown in Table 3, these preferences are translated into:* P_1*:* (m_3, m_1) *or*

$m_3 \succ_{P_1} m_1$, P_2: (m_4, m_5) or $m_4 \succ_{P_2} m_5$ and P_3: (m_4, m_1) or $m_4 \succ_{P_3} m_1$.

Moreover, tuples m_3 and m_4 can be seen as equally preferred whereas tuple m_2 is in an indifference preference relation with all other tuples in the database.

2.3 PREFERENCE COMPOSITION

Preferences affect one or more tuples in the database. When two different preferences act on the same set of tuples, different composition mechanisms can be used to determine a single preference relation.

One way to distinguish between different composition methods is based on *attitude* [40].

- Overriding attitude: one of the preference is given priority over the other and the second priority can be used only when the first one does not hold.
- Combinatory attitude: both preferences participate to the final ranking of the tuples.

Preference composition can be also classified as *qualitative composition* -when two qualitative preferences are combined -or *quantitative composition* -when two quantitative preferences are combined. In the *qualitative* case the final result is a pair-wise order of tuples, whereas in the *quantitative* case an aggregate score is attached to all affected tuples.

2.3.1 Quantitative Composition

In the quantitative case, an aggregation function is used in order to combine values assigned to a tuple by each preference.

Definition 6. *Quantitative preference combination function – Let P_1 and P_2 be two quantitative preferences over a relational schema R defined by the preference functions p_1 and p_2. Also, let F : $\mathbb{R}$ x $\mathbb{R} \rightarrow \mathbb{R}$ be the combining function. In this case, $\forall t_i, t_j \in$ R, $t_i \succ_{p_1 \text{x} p_2} t_j$ if and only if $F(p_1(t_i), p_2(t_i)) > F(p_1(t_j), p_2(t_j))$.*

Commonly used quantitative preference composition functions include weighted summation, average, minimum, and maximum.

Koutrika and Ioannidis [25] distinguish three different categories of quantitative preference compositions functions based on the final value assigned:

- *inflationary functions*: when the final score has a larger value than the two given values.
- *dominant functions*: when the final score is dominated by the score assigned by one preference.
- *reserved function*: when the final score lies between the two given preference scores.

2.3.2 Qualitative Composition

In the case of a qualitative composition we can define the mechanism in two different ways based on the attitude towards the preference domination.

In the first case, one of the preferences takes priority, as follows.

Definition 7. *Prioritized preferences composition – Given two preference relations, P_1 and P_2, over a relational schema R, the prioritized preference composition relation $\succ_{P_1 \& P_2}$ is defined over $\prod_{i=1}^{n} dom(A_i)$ x $\prod_{j=1}^{n} dom(A_j)$ such that $\forall t_i, t_j$ $\in$R, $t_i \succ_{P_1 \& P_2} t_j$ if and only if $(t_i \succ_{P_1} t_j) \vee (\neg(t_i \succ_{P_1} t_j) \wedge (t_i \succ_{P_2} t_j))$.*

In the second case, both preferences are considered equally important, as explained next.

Definition 8. *Pareto preference composition – Given two preference relations, P_1 and P_2 over a relational schema R, the pareto preference composition relation $\succ_{P_1 \otimes P_2}$ is defined over $\prod_{i=1}^{n} dom(A_i)$ x $\prod_{j=1}^{n} dom(A_j)$ such that $\forall t_i, t_j$ $\in$R, $t_i \succ_{P_1 \otimes P_2} t_j$ if and only if $(t_i \succ_{P_1} t_j \wedge \neg(t_j \succ_{P_2} t_i)) \vee (t_i \succ_{P_2} t_j \wedge \neg(t_j \succ_{P_1} t_i))$.*

Intuitively, under Pareto composition, a tuple is better than another if it is at least as good as the other one under one preference and strictly better under the other preference.

2.4 PREFERENCE GRAPHS AND PREFERENCE GRANULARITY

2.4.1 Preference Graph Definitions

A preference relation can be depicted as a directed graph, called a *preference graph*. The literature describes a few types of preference graphs, according to their purpose or the different levels of

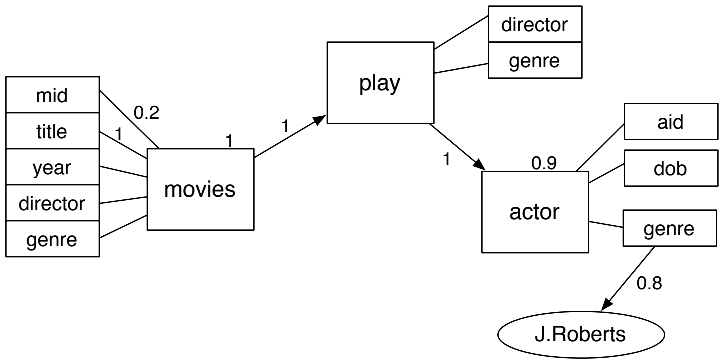


Figure 1: Personalization Graph Example as Presented in Pitoura et. al [40]

granularity a preference can be expressed. Stefanidis et. al. [40] give the following definition for a preference graph.

Definition 9. *Preference graph – A preference graph is a representation of a preference relation over an instance* r *of a relational schema* R. *In a preference graph, there is one node for each tuple* t *in* r *and there is a directed edge from the node representing tuple t_i to the node representing tuple t_j, if and only if $t_i \succ_P t_j$.*

Moreover, Koutrika and Ioannidis [24] introduce the notion of a preference graph – for expressing preferences over attributes – seen as an extension to the database schema graph, and defined as:

Definition 10. *Preference graph as schema extension – Let G=(V,E) be a directed graph (V is the set of nodes, E is the set of edges) that is an extension of the traditional schema graph. The set of nodes contains three types of nodes:*

- relation nodes, *one for each relation in the schema.*
- attribute nodes, *one for each attribute of each relation in the schema.*
- value nodes, *potentially one for each possible value of each attribute of each relation in the schema but only those that have any interest to a particular user are specified.*

Likewise, there are two types of edges in E:

- selection edges*, from an attribute node to a value node.*
- join edges*, from an attribute node to another attribute node.*

Moreover, a user's interest is expressed in the form of degree of interest*, which is a real number in the range [0,1]. 0 indicates lack of any interest, while 1 indicates extreme ('must-have') interest. These values are labels on the graph's edges.*

Figure 1 is an example of such graph; it contains preferences at different granularity levels. First, we have a quantitative preference described as a selection preference (i.e., <*actor.name*=*'J. Roberts'*, 0.8>). Second, we have a join preference (i.e., <*movie.mid* = *actor.aid*, 1> and <*play.aid* = *actor.aid*, 1>) which state that movies preferences are determined by actor preferences.

Aside from capturing a tuple-based or predicate-base preference relation, preference graphs have been also used to capture the relationship between different entities in the database [24] or the hierarchy of contextual preferences [35]. In the later case, a node in the contextual preference graph contains the preference along with the context where the preference holds. Moreover, it is assumed that a context is defined over a hierarchical domain and, therefore, an edge in this graph will capture this hierarchy.

Pitoura et.al. [35] have introduced the preference graph that focuses on preferences enhanced with context information. Context expresses conditions on situations external to the database or related to data stored in the database. Below we give the definition for this type of preference graph and in Figure 2 there is an example of such a graph, as presented in [35].

Definition 11. *Contextual preference graph – The preference graph PG_{Pr}=(V_{Pr}, E_{Pr}) of a profile Pr is a directed acyclic graph such that there is a node* v $\in$ V_{Pr} *for each context state* cs $\in$ *Context(Pr) and an edge $(v_i, v_j) \in E_{Pr}$, if the context state that corresponds to v_i is a tight cover of the context state that corresponds to v_j.*

Contextual preference graphs have been intensively studied in AI domain and they are known as conditional preference networks or *CP-nets* [6]. In the database domain limited work has been done on *hierarchical* CP-nets [32], *incomplete* CP-nets [10] and CP-nets translated into an expression in the formal preference language over strict partial orders [13].

Definition 12. *Conditional Preference Network – Let A=$\{A_1, A_2, \dots, A_d\}$ be a set of attributes. A Conditional Preference Network (i.e., CP-net) is defined over A as a directed graph in which there*

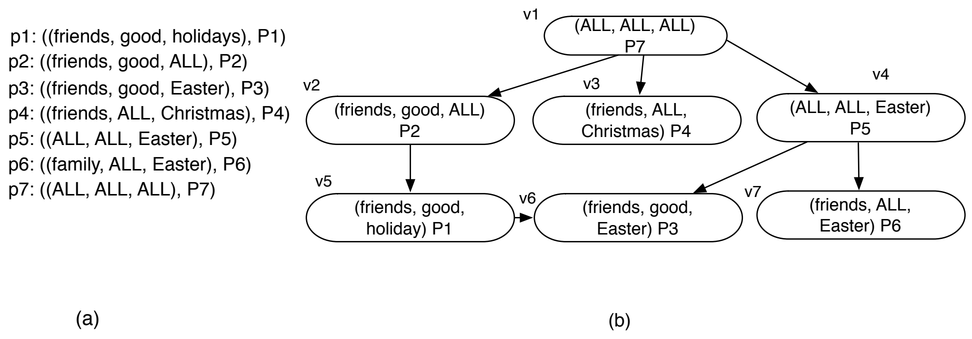

p1: ((friends, good, holidays), P1)
p2: ((friends, good, ALL), P2)
p3: ((friends, good, Easter), P3)
p4: ((friends, ALL, Christmas), P4)
p5: ((ALL, ALL, Easter), P5)
p6: ((family, ALL, Easter), P6)
p7: ((ALL, ALL, ALL), P7)
v1
(ALL, ALL, ALL) P7
v2
(friends, good, ALL) P2
v3
(friends, ALL, Christmas) P4
v4
(ALL, ALL, Easter) P5
v5
(friends, good, holiday) P1
v6
(friends, good, Easter) P3
v7
(friends, ALL, Easter) P6
(a)
(b)

Figure 2: Pitoura et. al [35] – Context Information Enhanced Personalization Graph

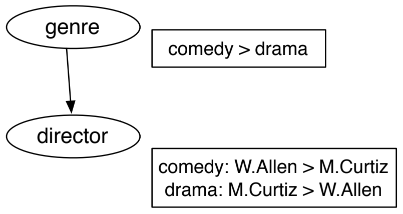


Figure 3: Pitoura et. al [40] – Personalization Graph Enhanced with Conditional Preference Tables

is a node for each attribute in A. If an edge from an attribute A_j to an attribute A_i exists in the graph, then A_j is an ancestor of A_i. Each node in the graph is annotated with a conditional preference table*, CPT, describing the preferences over A_i's values given a combination of its ancestors values.*

An example of this type of graph is given in Figure 3. Let Z_i be the set of all ancestors of A_i. Semantically, the preferences over A_i depend on the attributes Z_i. In this example, preferences over *director* depend on *genre*. For two tuples, a_1 and $a_2 \in dom(A_i)$, and a context $z_i \in dom(Z_1)$ – with A_i='director' and Z_i='genre' – we say that tuple a_1 is preferred over tuple a_2 under the context z_i. In Figure 3, *W. Allen* is preferred over *M. Cutiz* under the *comedy* context and *M. Curtiz* is preferred over W. Allen under the *drama* context.

2.4.2 Levels of Granularity for Preferences

In terms of granularity, a preference can be expressed at the tuple level, where each preference affects just a tuple. The following preferences are express at a tuple level:

- Quantitative preference: “I like the actress J. Roberts.”
- Qualitative preferences: “I like J. Roberts more than D. Moore.”

Attribute preferences express preferences between attributes. This type of preference can be used to rank tuples based on the priority of attributes involved in the preference relation. Alternatively, it can be used to capture priorities over preferences when composition techniques are applied. For example:

- Quantitative preference: “I prefer the director of a movie.”
- Qualitative preference: “I prefer the genre more than the duration of a movie.”

Relationship preferences are preferences based on the relationship of two entity types (i.e., generic relationship preference) or two particular entities (i.e., instance relationship preference). For example:

- General relationship: “A director directed a movie.”
- Instance relationship: “A particular actor played in a particular movie.”

2.5 CURRENT SUPPORT FOR HYBRID PREFERENCES AND PREFERENCE GRAPHS

Many solutions have been proposed for working with preferences [40]. In most of the cases the designed systems can handle only one type of preference (e.g., qualitative *or* quantitative). Our proposed model combines these two different approaches into a unified model. We propose a new type of graph that handles preferences and describes both qualitative and quantitative preferences with their associated intensity. To the best of our knowledge, there is no prior work that handles qualitative preferences in conjunction with quantitative preferences with the exception of Preference SQL [22] as mentioned in the Introduction.

The work done by Koutrika and Ioannidis [26] is the most related work to ours, which makes use of a preference graph. 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 ones to the least preferred.

The Preference SQL 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, when is connected with an ELSE operator to suggest that if the first criteria is not met, then the second one should be used. Also, in order to put a priority on preferences over different attributes, a PRIOR TO operator is provided. In this framework users need to fully describe their preferences for each query, in contrast to our approach, in which preferences are stored in user profiles and the system decides their applicability for each query. Because of this difference we refer to Preference SQL as a local approach and ours as a global one. To illustrate the difference between these approaches, consider the following example with a snapshot of the dealership relation given in the Table 5.

Example 5. *Assume the three preferences over car entities:*

P_1: *"I like a car with the price between \$7,000 and \$16,000".*

P_2: *"I prefer car with a mileage between 20,000 and 50,000".*

P_3 : *"I prefer a* BMW *or a* Honda*".*

Also, we know that preference P_2 is more important than preference P_3.

In Preference SQL we have two ways to write this preference, in the PREFERRING clause. First way is to state that all preferences are equally important, which results in the following preferring clause: *PREFERRING (price between 7000 AND 16000) AND (mileage between 20000 and 50000) AND make IN ('BMW', 'Honda') Top 3*, where Top 3 specifies how many tuples to return.

Another way is to consider one preference more important than the other. For example, as this

Table 5: Dealership Relation

id	price	mileage	make
t1	$7,000	43,489	Honda
t2	$16,000	35,334	VW
t3	$20,000	49,119	Honda

example states, the preference on *mileage* should be more important than the preference on *make*. In this case the preferring clause has the following form: *PREFERRING price between 7000 AND 16000 AND mileage between 20000 and 50000 PRIOR TO (make IN (‘BMW’, ‘Honda’).* In both cases, the results returned are: t1, t3 and t2, in this order.

However when submitting this query, we expect {t2} to be the second most preferred tuple because tuple t2 matches the preference on *price* and *mileage* and the preference on *make* is not of high priority. On the other hand, tuple t3 does not match the preference on *price*. As explained in the previous sections, the meaning of intensity score in a qualitative preference is to express the strength of the relationship between two basic preferences, in this case preference on *mileage* and preference on *make*. The user specified that *the mileage* of the car is more important than *the make*. Although Preference SQL can attach a score to each tuple, this score will refer to that particular tuple in respect to *all* other tuples in the database and, therefore, will not be able to capture the connection between two tuples as a qualitative preference does. Our system can rank tuple t1 as the first tuple, followed by tuple t2 and t3 as explained later, in Section 4.6.1, which is the expected result.

2.6 SUMMARY

This section overviewed the theoretical background of our work that describes the different types of preferences and preference graphs defined in the literature. Moreover, we discussed Preference SQL, which is the only system that works on combining qualitative and quantitative preferences but with a semantical limitation and the lack of a user profile. In the next chapter we are introducing our new model that overcomes Preference SQL limitations and combines qualitative and quantitative preferences in the same graph.

3.0 UNIFIED PREFERENCE GRAPH MODEL

In the previous chapter we defined database preferences and we described the state of the art in terms of preference representation and query personalization. We also emphasized the positive and negative aspects of both types of preferences in Table 2. In the following chapter we present our hybrid preference model, which is formalized using a labeled and acyclic graph representation and examples, followed by a discussion about how this model can be used and improved.

3.1 WHY IS A UNIFIED DATABASE PREFERENCES MODEL NECESSARY ?

We argue that is necessary to have a hybrid model that supports both types of preferences along with their intensity value because such a system will be able to support all possible preferences, rank tuples from the most preferred to the least preferred based on the total order created, and store, along with the predicate (or set of predicates), the strength of each preference.

More specific, we need a hybrid model because:

(1) The qualitative preference approach is the most general model that can support all types of preferences a user may wish to specify.

(2) The quantitative preference approach returns a total order of the tuples that match a user's preferences.

(3) When the intensity of a preference is not captured, an unexpected order of the tuples may be returned.

3.2 UNIFIED MODEL FOR PREFERENCES

As discussed in Chapter 2, a graph representation is the most natural way of exemplifying the connections between tuples in a database and visually depicting their relationships. The purpose of our preference graph is to connect two different preference approaches into a unified model, and to record the strength/intensity of each preference for future use.

Definition 13. *Preference Intensity –* **Intensity** *represents the strength of a preference and can be expressed as a decimal point value between* -1 *and 1 with the following meaning:*

1. *All negative values are used to express negative preferences,* -1 *being used to express complete dislike.*
2. *All positive values are used to express positive preferences and 1 is used to capture the most preferred tuple.*
3. Zero *is a special value used to express* $\underline{\textit{equally preferred}}$ *tuples, in the case of qualitative preferences, and* $\underline{\textit{indifference}}$*, in the case of quantitative preferences.*

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]).

HYPRE (read [‘haipə]) Graph, is our proposed hybrid graph model defined to support both qualitative and quantitative preferences along with their intensity value, if any value is provided.

Definition 14. *Hybrid Preference (HYPRE) Graph – We define a hybrid preference graph PG =(PV,PE) as a labeled directed and acyclic graph where:*

1. *PV is a set of vertices where each vertex represents a tuple in the database or a query predicate.*
2. *PE is a set of edges where each edge (v_i, v_j, s), v_i, v_j ∈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.*
3. *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.

3.2.1 Tuple-based vs. Predicate-based Preference Graph

A vertex in the graph can represent a single tuple in the database (*tuple preference graph*), or a set of tuples if it is defined as a query predicate (*predicate preference graph*).

A tuple-based preference graph holds a single tuple in each vertex. Because of that, this preference graph can be seen as a materialized view and there is a direct access to the intensity value of each tuple. However, a tuple-based preference graph is usually not scalable. For each tuple that matches a preference, a new vertex needs to be created in the preference graph. However, this type of preference graph can be seen as a materialized database view and is useful especially in cases when the preference has a low probability of changing.

A predicate-based preference graph is a scalable version of the tuple-based one, since a vertex matches multiple tuples, and it is used for preferences that apply to a large set of tuples. This type of preference graph is also useful for preferences that are removed and reinserted often because it is much easier and efficient to add/remove only one vertex in the graph rather than all vertices that represent tuples that match a particular preference. However, queries enhanced with one or more predicate need to be run to determine the tuples that match them and to assign an intensity to these tuples. This process is time consuming, comparing with the tuple-based model. Nonetheless, the scalability aspect of the model and the fact that the time overhead introduced by the query enhancement and running is not significant, makes this model an ideal way to store preferences.

From the representation point of view, both types of graphs are similar. Moreover, the predicate-based graph representation subsumes the tuple-based graph representation because we can always define a predicate that only return one tuple. For of this reason, we only focus on the predicate-based preference graph in the rest of this dissertation.

3.2.2 Attribute-based Preference Graph

In addition to *tuple-based* and *predicate-based*, an attribute-based preference graph can be defined. Each type of graph differs from the other on the information hold in each vertex. In the attribute-based preference graph, each vertex holds a preference on an attribute, rather than an attribute value. Even though an attribute-based graph is not implemented in this work, any tuple-based and predicate-based preference graph can be easily extended to include attribute-based vertices. However, applying the attribute-based predicates to a user-submitted query necessitates more work to translate the selected attribute and the associated function into an SQL predicate.

A preference defined on an attribute should be accompanied by a function that determines the order of tuples. For example, an attribute-based preference can be : "When I search for hotels, I prefer the price". The preferred attribute in this case is *price* but this preference is not complete if the user does not specify the function to be applied over this attribute. A complete attribute-based preference is: "When I search for hotels, I prefer the cheaper rooms". In this case, the attribute is *price* and the function is *minimum*.

3.3 SPECIFYING PREFERENCES IN HYPRE GRAPH

Based on the definition presented in the previous section, we can now illustrate how one can create a preference graph to support different types of preferences. We are using an instance of the DBLP-Citation-network V4 dataset [44] (see Table 6) and we show how different types of preferences can be represented in the same preference graph. We start with an empty preference graph and we incrementally add new preferences in this graph, one preference type at a time.

We organize the presentation of the examples using the two main types of preferences:

- quantitative preferences – used to express a positive or negative preference towards an item
- qualitative preferences – involving two different nodes

Table 6: The DBLP Relation

pid	Title	Year	Venue
t1	Automated Selection of Materialized Views and Indexes in SQL Databases	2000	VLDB
t2	Composite Subset Measures	2006	VLDB
t3	Keymantic: Semantic Keyword-based Searching in Data Integration Systems	2010	PVLDB
t4	Proximity Rank Join	2010	PVLDB
t5	iNextCube: Information Network-Enhanced Text Cube	2009	PVLDB
t6	Processing Proximity Relations in Road Networks	2010	SIGMOD
t7	Relational Joins on Graphics Processors	2008	SIGMOD
t8	Refresh: Weak Privacy Model for RFID Systems	2010	INFOCOM
t9	Congestion Control in Distributed Media Streaming	2007	INFOCOM

3.3.1 Quantitative Preferences in the HYPRE Graph

Let PV be the set of nodes, and PE the set of edges in the preference graph. Initially, the preference graph is empty therefore PV= $\emptyset$ and PV= $\emptyset$.

Assume we have the following four preferences.

- P1: *"I prefer papers published between 2000 and 2005, with intensity 0.3"*
- P2: *"I prefer papers published between 2005 and 2009, with intensity 0.5"*
- P3: *"I like papers published after 2009 with intensity 0.8"*

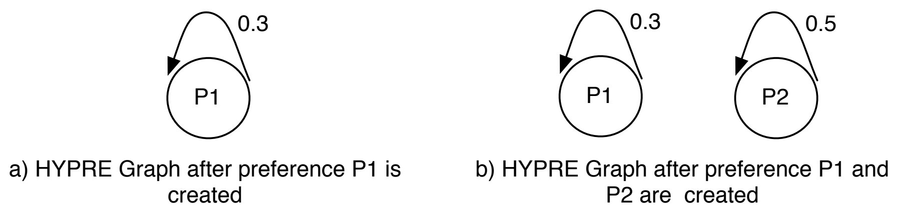

0.3
P1
a) HYPRE Graph after preference P1 is created
0.3
P1
0.5
P2
b) HYPRE Graph after preference P1 and P2 are created

Figure 4: Quantitative Preferences

All these preferences are examples of quantitative preferences, with different level of intensity values. They refer to a set of tuples, out of all the tuples in the database and can be translated in the following preference nodes:

– P1 $\in$ PV: $\{$ the predicate =“year$\geq$2000 AND year$\leq$2005” $\}$, and e1 =(P1, P1, 0.3) $\in$ PE.

– P2 $\in$ PV: $\{$ predicate=“year$\geq$2005 AND year$\leq$2009” $\}$ and e2 =(P2, P2, 0.5) $\in$ PE.

– P3 $\in$ PV: $\{$ the predicate=“year$\geq$2009” $\}$ and e3 =(P3, P3, 0.8) $\in$ PE .

In order to add these preferences in the graph we will need to create a node for each preference defined above. Since initially the graph is empty, we do not need to verify if a node with the same predicate is already inserted in the graph.

Fig.4 shows the state of the graph after (a) preference node P1 is inserted and (b) preference node P2 is inserted in the graph.

Also, assume the following preference:

- P4 (Negative Preference): *“I am not interested in papers published in INFOCOM.”*

This is an example of a negative preference and it will be translated in the following preference node:

– P4 $\in$ PV: $\{$ predicate=“venue=INFOCOM” $\}$ and e4=(P4, P4, -1) $\in$ PE.

All these preferences are examples of quantitative preferences and a node for each preference is created in the preference graph. Fig 5 shows the entire graph, after all these preferences are inserted in the preference graph. For the negative preference we can see that only tuples p8 and p9

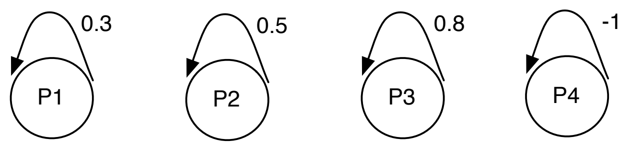


Figure 5: HYPRE Graph After All Quantitative Preferences are Inserted

match predicate P4 and there are no tuples matching preference P3. However, if more values are added into the database, the preference graph does not need any modification.

3.3.2 Qualitative Preferences in the HYPRE Graph

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 are not already part of the graph, we create the necessary node(s) and connect them with a directed edge.

Qualitative preferences can express multiple forms of preferences as illustrated in the following examples.

- Relative Preferences: *"If two papers are published in VLDB, I prefer, with intensity 0.8, the one published in the last 4 years."*

  A *relative preference* relates two tuples or two disjoint sets of tuples belonging to the same set. For this preference we create two different nodes:

  - P5 $\in$ PV: { predicate = "venue=VLDB AND year$\geq$2010"} and e5 =(P5, P5, N/A) $\in$ PE
  - P6 $\in$ PV: { predicate = "venue=VLDB AND year$<$2010"} and e6 =(P6, P6, N/A) $\in$ PE

  and an additional edge:

  - e7 = (P5, P6, 0.8) $\in$ PE.

  The qualitative and quantitative cases are differentiating in two aspects:

  1. The nodes created in the qualitative case do not have an intensity value initially assigned, unless the node was already defined in the graph as a quantitative preference

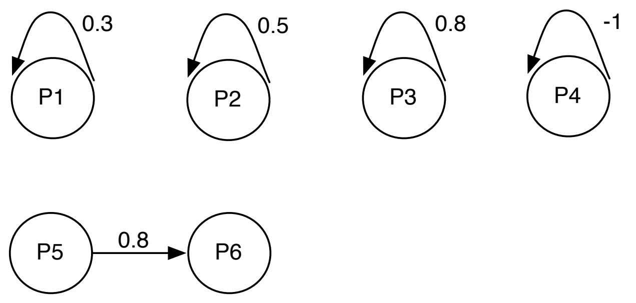


Figure 6: HYPRE Graph with Relative Preference

2. Inserting a qualitative preference implies inserting an extra directed edge co connect the two nodes that make the qualitative preference.

Fig. 6 is the new HIPI Graph, after this qualitative preference is inserted.

- Preference Set: *"From a list of papers, I prefer papers published in VLDB and as many papers as possible published after 2009."*

  A *preference set* relates two or more not necessarily disjoint set of tuples, from which one is a subset of the other.

  This preference can be seen as a qualitative preference with *papers published in VLDB* slightly more preferred than *papers published after 2009*. Using the qualitative preference definition, the comparative expression *slightly more preferred* can be translated into a small intensity value for the entire qualitative preference (e.g., 0.2). Moreover, we can see that a node having the predicate="year$\geq$2009" is already defined in out graph (i.e., node P3) therefore there is no need to create a new node for this preference. We only need to use this node in the new qualitative preference.

  With this in mind, we can now formally define the new preference:

  – P7 $\in$ PV: { predicate = "venue=VLDB"} ; e8 =(P7, P7, N/A) $\in$ PE

  – e9 = (P7, P3, 0.2) $\in$ PE

  The graph representation is given in Fig 7.

- 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"*

  The *different levels of intensity* relates two disjoint tuples or set of tuples and the intensity value captures how much more is one preferred over the other.

  In the preference graph, we already have node P7 with the {predicate = "venue=VLDB"}. We are going to use this node for the left part of the qualitative preference. For the right part we need to create a new node and a new edge to actually create the qualitative preference:

  – P8 $\in$ PV: { predicate ="venue = SIGMOD" }; e10 =(P8, P8, 0.8) $\in$ PE.

  – e11 = (P7, P8, 0.3) $\in$ PE. As in the previous example, we map the expression "a bit more" to a small intensity value (e.g., 0.3).

3.4 SUMMARY

This chapter introduced formally our hybrid preference model. We defined two possible types of preference graphs – tuple-based and predicate-based preference graphs – and illustrate how one can create a preference graph to support different types of preferences. Predicate-based model subsumes the tuple-based model and there is no change between the two from the presentation point of view. In the tuple-based case we need to replace the nodes that contain a predicate with

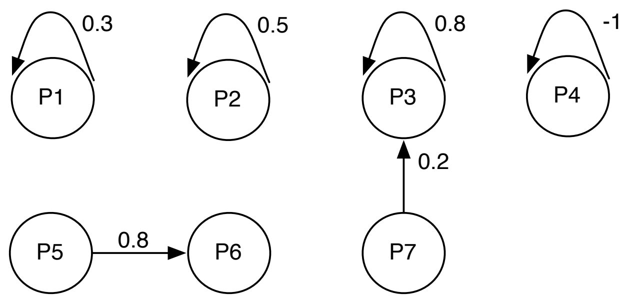


Figure 7: HYPRE Graph with Relative Preference and Set Preference

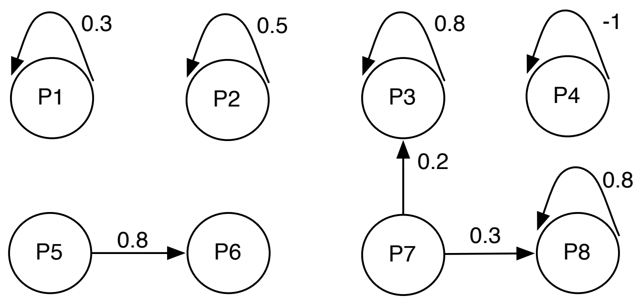


Figure 8: Final Version of HYPRE Graph

nodes that only contain a tuple, or a set of tuples. However, when the database changes, the tuple-based preference graph needs to change too, which will require a lot of time overhead, as we expect the data to change in a real situation (e.g., more papers are added in the DBLP dataset; new information is added for the incomplete entries; year published is updated for the wrong entries).

4.0 HYPRE GRAPH – FROM THEORY TO IMPLEMENTATION

In terms of representation, the HYPRE Graph, introduced in the previous chapter, is sufficient to store for each user, their list of preferences. However, we also need to be able to easily traverse the graph, find paths between nodes, order nodes in terms of their intensity values, and detect nodes that represent the same preference. In this chapter, after examining different alternatives, we propose a graph-based implementation of HYPRE which stores both qualitative and quantitative database preferences in the same graph, as SQL predicates, along with the intensity or the strength of each preference.

4.1 GRAPH REPRESENTATIONS

The most common representation for graphs is an *adjacency matrix*. For each new node introduced in the graph, we need to create one row and one column in the associated adjacency matrix. Moreover, each cell (i,j) contains a value that expresses if there is an edge between node *i* and node *j*. For our preference graph, in order to use this representation we need to create one row and one column for each preference and we record the intensity value in the adjacency matrix. For each cell (i,j), if $i \neq j$ then the value represents the intensity of a qualitative preference, whereas, if $i = j$, the value stored in cell (i,i) represents the intensity of a quantitative preference. For any two nodes in the graph, if there is no edge connecting them then we use the *NULL* value in the corresponding cell in the adjacency matrix or an empty cell. Figure 9 shows an example of such graph, and its associated adjacency matrix. In this representation, m1 $\ldots$ m5 represents either a single tuple or an SQL predicate.

Although there are a number of solutions suited for graphs represented as adjacency matrices

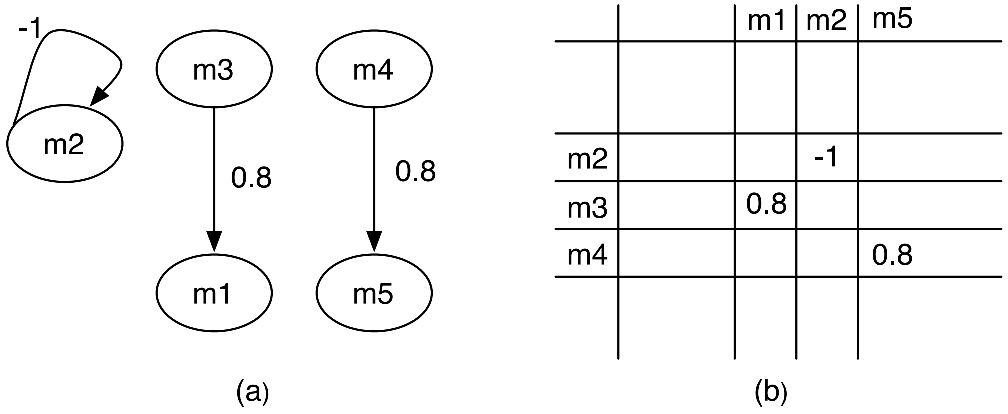


Figure 9: Relative Preferences Specifications (a) Graph Representation; (b) Adjacency Matrix

and many optimized algorithms to traverse these graphs have been implemented, in our case such solutions would not be efficient. First, we would need to store an $m \times m$ matrix, with m being the number of preferences for one particular user, although we expect that a node would be connected with at most k nodes, where k is much smaller than m. Because of that, most of the cells would contain *NULL* values, i.e., the adjacency matrix would be quite sparse. Second, we need to have an additional data structure to hold the actual preference predicates, since the associated matrix will only contain information on the intensity values based on the node id and not the value of the predicate. Moreover, every time we introduce a new node we need to perform a search on the preference storing structure to make sure the node does not already exist, which implies that we need to build an index structure. Finally, since we are building user profiles, we would need to store many preference graphs, one for each user (i.e., many adjacency matrices).

Since each preference in our graph is a predicate and each predicate will be used to enhance an SQL query at query time, another way to store the preference graph will be in a *relational database*. We can create two tables for preferences, qualitative (see Figure 11) and quantitative (see Figure 10). With an index on the preference value in the quantitative table we can guarantee fast searches on preexisting preference predicates and there will be no space overhead, since we only store the preference predicates. Also, for each preference we can store the user_id which allows

pfid	uid	preference	intensity
1	1	dblp.venue="SIGMOD Conference"	0.476190476190476
2	1	dblp.venue="ICDE"	0.238095238095238
3	1	dblp.venue="Modern Database Systems"	0.0952380952380952
4	1	dblp.venue="VLDB"	0.0952380952380952
5	1	dblp.venue="Inf. Syst."	0.0952380952380952
6	2	dblp.venue="SIGMOD Conference"	0.285714285714286
7	2	dblp.venue="INFOCOM"	0.238095238095238

Figure 10: Quantitative Preference Storage

pfid	uid	leftPref	rightPref	intensity
1	1	dblp_author.aid=3706	dblp_author.aid=6	0.0098
8	1	dblp_author.aid=61511	dblp_author.aid=8	0.0097
15	1	dblp_author.aid=617772	dblp_author.aid=10	0.0097
19	1	dblp_author.aid=4103	dblp_author.aid=12	0.0097
44	1	dblp_author.aid=1714100	dblp_author.aid=9	0.0097
194	2	dblp_author.aid=169	dblp_author.aid=788	0.0288
195	2	dblp_author.aid=788	dblp_author.aid=11	0.0206

Figure 11: Qualitative Preference Storage

us to keep together all users' profiles. However, we still need to find efficient ways to traverse the graph, find paths and cycles, which is well-known to be very inefficient within traditional relational database system since it involves recursion. Typically, a recursive join is implemented using adjacency matrices or a graph structure.

The third solution, and the most efficient way to store preferences that will eliminate the overhead introduced by the graph traversals and search is, after all, a graph representation where each node represents a quantitative preference characterized by different attributes as the preference predicate, the intensity value and any other necessary information. Moreover, an edge in the graph will connect two nodes and, therefore, will create a qualitative preference.

4.2 HYPRE REPRESENTATION

Our solution to implement a hybrid preference graph is to use a *graph database system* which is designed to provide efficient graph traversal and graph manipulation.

Node: In this implementation, a node contains three properties : *(user_id, predicate, intensity_value)*, where intensity_value is the quantitative intensity. Moreover, a node can have one ore more labels attached, depending on the what metadata is necessary (e.g., label used for indexes, label for intensity value provenance).

Edge: An edge in this graph needs to store only one property, the intensity value, to capture a qualitative preference intensity. Moreover, we use a label for each edge in the graph, and there are three different edge types, classified based on this label. The most important one is PREFERS, which specifies a valid qualitative preference. 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.

This way, we can easily create only one graph and, using the *user_id* property of a node, select all the nodes for a particular user, as needed.

4.3 NEO4J – A SCALABLE GRAPH DATABASE SYSTEM

We decided to use Neo4j[1], a graph database engine to store the preference graph. All preferences and their associated intensity values, for all users, are stored in one single graph. Each node in the graph has four properties:

- **node_id**: an internal id, automatically generated by the graph database when the node is created,
- **user_id**: the id of the user for whom the preference is created,
- **predicate**: the SQL predicate that represents a preference, and
- **intensity**: the intensity value of the preference.

Fig. 12 shows, for the node with *node_id*=33, the values of the properties described above, as returned when the Neo4j graph database is queried.

Scalability. Neo4j is a scalable graph database that can be used both as a server or embedded in different programing languages. In our stress tests we were able to insert 7 billion nodes before we intentionally stopped the node insertion process. The test was set up to insert 1 million tuples at a time and report back the time necessary to do the insertion. As expected, the more nodes we have in the graph, the longer it takes to do the insertion. However, even when we inserted the

[1]http://www.neo4j.org/

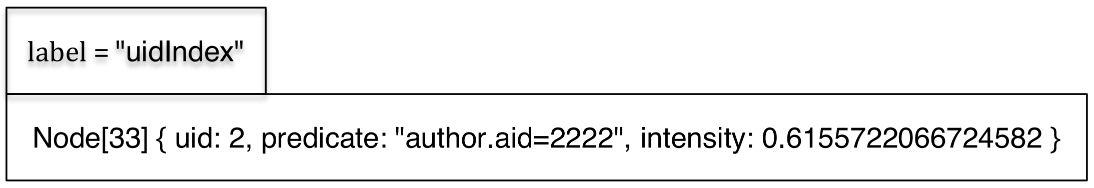

label = "uidIndex"
Node[33] { uid: 2, predicate: "author.aid=2222", intensity: 0.6155722066724582 }

Figure 12: Example of a Vertex in the HYPRE Graph

last million tuples, the system needed less than 70 sec to finish the insertion. These results are presented in Fig. 13.

Efficient Search. When dealing with user profiles, we need to be able to quickly retrieve all the preferences for one particular user. For example, to retrieve all nodes for user_id=2, with the intensity value greater than zero, without any indexes, the system needs around 3600 sec, which is very inefficient if it is to be implemented in an interactive system. However, Neo4j provides indexing which we utilize to support interactive search.

The best indexing scheme, with the best retrieval time, is based on the label of the node and the value of one property. When searching for preferences, we are interested in all nodes created for one particular user. Because of that we indexed the graph using the *user_id* property. In addition, since all nodes in our graph are preference nodes, we marked all of them with the label *uidIndex* and created the index *uidIndex(uid)*. With this indexing scheme, the system needs less than one second to return all the nodes for user_id = 2 (i.e., down from 60 sec for the case with a simple index).

Cypher. In order to create, update and query the database graph we use CYPHER, a declarative query language for Neo4j [2]. CYPHER is the Neo4j's query language that allows fast retrieval of nodes along a path, with a particular property or with a specified label. The select query structure in CYPHER has the following general form:

START [MATCH] [WHERE]

RETURN [ORDER BY] [SKIP] [LIMIT];

The START clause specifies the starting point. This can be a single node identified by its id, multiple nodes, all nodes (e.g., n=node(*)) or nodes specified by a parameter that will be substi-

[2]http://www.neo4j.org/learn/cypher

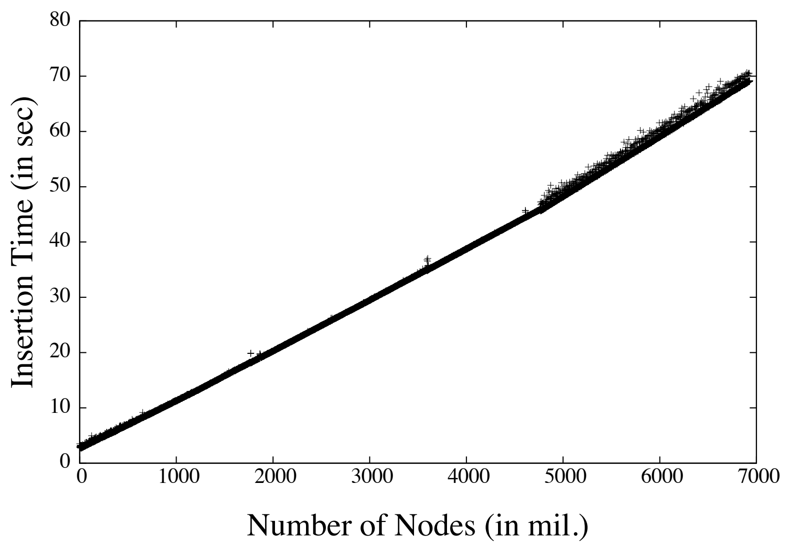


Figure 13: Node Insertion Time for 7 Billion Nodes, in 1 Million Batch Size.

tuted with a given value at the query time in an application.

The MATCH clause specifies what type of relationships (i.e., edges) should be traversed. As explained earlier, in our graph, there are three possible edges: PREFERS, CYCLE and DISCARD.

As an example of how a MATCH clause can be used to find all qualitative preferences that start at a particular node (i.e., all nodes that are connected with an edge of type PREFERS to a particular node) we use the following CYPHER query:

START n = node(id)

MATCH n -[:PREFERS] → m

RETURN id(n) as leftId, id(m) as rightId;

The WHERE clause introduces additional selection constrains. For example, when we query the graph we want to return nodes for a particular user only. Because of that, we need to specify the value for the *user_id* property in the WHERE clause of the query. The following example returns a list of all preferences and their associated intensity value, for one particular user, in descending order, based on their intensity values.

```
START n=node(*)
WHERE n.uid=uid
RETURN n.preference, n.intensity
ORDER BY n.intensity desc;
```

Using CYPHER we query the graph database to return, based on the user's preferences, a list of SQL predicates in descending intensity order, excluding preferences with negative values. This list will then be used to enhance a user-provided query and account for the user's preferences.

4.4 ESSENTIAL FUNCTIONS TO (RE-)COMPUTE INTENSITY VALUES

The mechanism of computing/recomputing the intensity value is one key aspect of our unified model. Being able to assign intensity values to nodes that do not have one given (i.e., qualitative preferences only have an intensity assigned to the edge) allows us to increase the number of quantitative preferences and, therefore, increase the "coverage" over all data of interest to the user. As a reminder, quantitative preferences allow for a total order and a qualitative preference in our model, can be seen as two quantitative preferences (one stored in the left node and one stored in the right node) connected by a directed edge, from the left node to the right node. The directed edge – from left to right – expresses the preference: *Tuples returned by the preference stored in the left node are more preferred than tuples returned by the preference stored in the right node*.

We defined two functions, in Equation (4.1) and Equation (4.2), to compute a meaningful intensity value, based on the intensity value or strength of the qualitative preference and one existing quantitative preference intensity value. *Intensity_Left* (Equation (4.1)) computes the value of the intensity for the *Left* node, and *Intensity_Right* (Equation (4.2)) computes the intensity value of the *Right* node.

$$\text{Intensity\_Left (Left, ql, qt)} = \min(1, \text{qt} * 2^{\text{sign(qt)*ql}}) \tag{4.1}$$

$$\text{Intensity\_Right (Right, ql, qt)} = \max(-1, \text{qt} * 2^{\text{-sign(qt)*ql}}) \tag{4.2}$$

The functions have the following parameters:

1. the position of the node – *left* or *right*
2. the qualitative preference's intensity value – *ql*
3. one quantitative preference's intensity value – *qt*.

The pair of functions defined in Equation (4.1) and Equation (4.2) are one example of such functions. However, any other function that conserve the graph model characteristics can be used. Any function defined with this purpose in mind should have the following properties:

1. When the function is used to compute an intensity value for the left node, it should return a value greater or equal to the intensity value in the right node.
2. When the function is used to compute an intensity value for the right node, it should return a value smaller or equal to the intensity value in the right node.

   The first two properties comes from the definition of a qualitative preference in our preference graph – *a directed edge from the left node to the right node*. This definition implies that the intensity value in the left node is greater or equal to intensity the value in the right node.
3. The new intensity value should be proportional to the strength of the qualitative preference. If the intensity value for the qualitative preference is *zero* then the two preferences are equally preferred and the value returned by the function should be equal to the intensity of the right node. However, if the qualitative preference's intensity is very large, the value returned should be much larger/smaller than the given quantitative preference's intensity in order to capture the meaning of the qualitative preference's intensity – the higher the value, the stronger the preference, and the higher the difference between the two quantitative preferences.
4. The function should not return values outside the range [-1, 1].

The -1/1 as upper-bound for *Intensity_Left* and *Intensity_Right* functions respectively are also not the only possible bounds. Actually, in practice the upper-bound should be a value smaller than the minimum/maximum value, since these are very strong negative/positive opinions and, as the value is computed by the system, we want to avoid assigning the highest possible value except when is explicitly assigned by one user.

This functions are applied every time a qualitative preference is created in order to compute a new value, recompute an old one to preserve the graphs' properties, or detect a conflict.

- Compute a new value. A new intensity value is computed in one of the following two cases:

  - A new qualitative preference is inserted in the graph. For this preference, two new nodes and an edge connecting them are created. The qualitative preference comes with an intensity value that is assigned to the edge and which represents how much more important one preference is over the other, but it does not indicate anything about each preference (i.e., each node), taken separately. In this case, a default value is assigned to one of the two nodes and a new value is computed for the other node.

  - A new qualitative preference is inserted in the graph, but this preference contains only one new node (with no intensity value) connected with an existing node. In this case, we compute an intensity value for the node that does not have an intensity value assigned yet, using the intensity value of the node already in the graph and the intensity value of the qualitative preference.

- Recompute a value. When the two nodes involved in a qualitative preference are already in the graph, a directed edge is created between them and a conflict check routine is executed. When a conflict of incompatible values is detected (i.e., the intensity value in the left node if lower then the intensity value in the right node) one of the two values needs to be recomputed. In order to avoid a conflict propagation, we recompute the value for the node that has no other connection in the graph, except for the newly introduced edge. There are two possible situations: the node selected has either in_degree=0 and out_degree=1 or in_degree=1 and out_degree=0. In Figure 14 we recompute the value for the node P1 and in Figure 15 we recompute the value for the node P2.

- Mark a conflict. If both nodes are already connected to the graph, the new edge inserted to create a qualitative preference will increase the value of in_degree or out_degree in which case both these values become greater than zero. In this case, the new edge is labeled with PREFERS only if it does not create an incompatible values conflict. Otherwise it is still inserted, but label it as DISCARD.

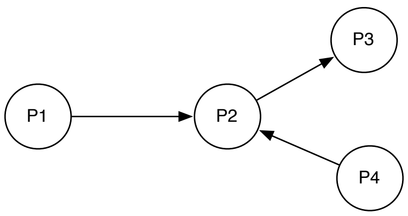


Figure 14: In_degree=0 for Node P1

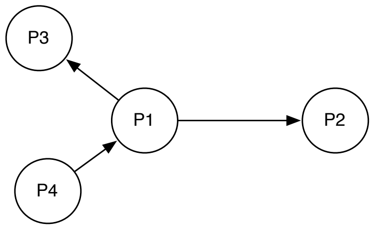


Figure 15: Out_degree=0 for Node P2

4.5 ALGORITHM FOR HYPRE GRAPH GENERATION

As mentioned in the previous section, we store preferences in a graph, where a node is defined by (*node_id, user_id, predicate, intensity*) that records all necessary information about each preference. For one particular user, the user's preference profile is represented by the subgraph generated using the subsets of all nodes with a particular *user_id*, and all edges that connect these nodes.

A node with no connection in graph represents a quantitative preference. Two connected nodes creates a qualitative preference with the direction of the edge to define the prefer order between predicates. This implies that, if intensity values of these nodes exists, then the value of the node that has an outgoing edge (refered to as the *left node*) must always have a greater or equal intensity value with respect to the node where the edge ends (refered to as the *right node*).

Moreover, recall that each edge has an associated 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.

The algorithm for creating the graph works incrementally.

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

Step 2. We add all qualitative preferences. For this case, there are two possible situations:

- if the nodes are already in the graph, the algorithm adds a directed link between them and, eventually, recomputes the intensity values, if needed
- if one or both nodes for one particular preference are not yet in the graph, the algorithm creates the nodes, assigns default values and recomputes the intensity value for only one node given the default intensity value and the intensity value of the qualitative preference.

Algorithm 1 shows the necessary steps taken to insert preferences into the preference graph. The algorithm will create a node in the graph only if it does not exist already (i.e., there is no node with the same user id and the same preference value). If the node is already in the graph then the *createOrReturnNodeId()* function returns the id of that particular node and does not create another node.

4.6 PREFERENCE AWARE QUERY ENHANCEMENT

Assume that a user with *uid*=2, and the user profile defined in Table 7, has submitted the following query: *"Show me all papers from the DBLP database"*. Without any knowledge of preferences, the query is submitted to the Relational Database Management System as:

```
SELECT * FROM dblp;
```

However, this query will return all 1,600,000 tuples, in a no "interesting" order for the user. This situation is known as the *information flooding* problem. Any system working with large datasets needs to alleviate this problem.

Instead of returning a list with all papers, in no particular order, we rewrite the query to enhance it with preferences from the user profile as:

```
SELECT * FROM dblp
WHERE <Combined list of preference>;
```

Table 7 displays a snapshot of the preferences, stored in the preference graph, for our user.

Algorithm 1 Create Preference Graph

Input: *A partial graph containing all quantitative preferences and a list of qualitative preferences.*

Output: *A complete preference graph with both, quantitative and qualitative preferences.*

```
1: BEGIN
2: addAllQuantitative();
3: for (each qualitative preference QL) do
4:     leftNodeId =createOrReturnNodeId(QL.leftPref);
5:     rightNodeId =createOrReturnNodeId(QL.rightPref);
6:     if (there is a path from rightNodeId TO leftNodeId) then
7:         createEdge(leftNodeId, rightNodeId, QL.intensity, CYCLE);
8:     else if ( degree(leftNodeId)=degree(rightNodeId)=0 AND conflict(leftNode, rightNode) ==
       FALSE ) then
9:         createEdge(leftNodeId, rightNodeId, QL.intensity, PREFERS);
10:        leftNode.intensity =computeIntensity(LEFT, QL.intensity, rightNode.intensity);
11:    else if ( degree(leftNodeId) $>$ 0 AND degree(rightNodeId) = 0 AND conflict(leftNode,
       rightNode) == FALSE ) then
12:        createEdge(leftNodeId, rightNodeId, QL.intensity, PREFERS)
13:        rightNode.intensity =computeIntensity(RIGHT, QL.intensity, leftNode.intensity);
14:    else if ( degree(leftNodeId) = 0 AND degree(rightNodeId) $>$ 0 AND conflict(leftNode,
       rightNode) == FALSE ) then
15:        createEdge(leftNodeId, rightNodeId, QL.intensity, PREFERS);
16:        leftNode.intensity =computeIntensity(LEFT, QL.intensity, rightNode.intensity);
17:    else
18:        createEdge(leftNodeId, rightNodeId, QL.intensity, DISCARD);
19:    end if
20: end for
21: END
```

In Chapter 6 we discuss how such preferences are extracted from DBLP dataset. We can see that there are two preferences that refer to the venue where a paper was published and two preferences related to the author of a paper.

Table 7: List of Preferences for uid=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

There are three obvious ways to combine these preferences:

1. Connect all preferences using the AND operator (conjunctive clause)
2. Connect all preferences using the OR operator (disjunctive clause)
3. Connect some preferences using AND and some preferences using OR operator (mixed clause)

Ideally we would like to return tuples that match all preferences, therefore connecting all preferences with an AND operator should be the preferred case. However, this is not always possible and the user profile from Table 7 illustrates this situation (e.g., it is not possible to have a paper that was published in two different venues[3]). This is the second known problem with traditional queries – the *information starvation* problem. The starvation problem appears when no tuples are returned because the condition is too selective and all tuples are filtered out.

Connecting all the preferences with an OR semantics brings more tuples in the result list, therefore avoiding the starvation problem. But this might also return all the tuples, if the user's preferences cover all the tuples in the database, in which case this is not a satisfactory solution either.

In order to avoid an empty result, we combine preferences that are referring to the same attribute with OR semantics. In order to be inclusive, we connect preferences that are defined over different attributes with AND semantics. Using this rule, we can now enhance the provided query with the preferences defined for uid=2, above. The rewritten final query has the following form:

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

The above idea clearly generalizes for all predicates and users in our system.

4.6.1 Preference Combination and the Combined Intensity Value

Once created, the preference graph is used to identify the relevant preferences and enhance the user-submitted query. As discussed above, we have adopted a mixed clause scheme to combine preferences, to verify the behavior of combined intensity value and the size of the resulted list, and a conjunctive clause scheme to find the most preferred tuples.

Stefanidis et. al [40] 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

[3]Of course we assume here that both the DBLP data is accurate and that standard academic ethics apply, so that no author is doing blatant "double-dipping" by publishing the exact same paper in two different venue!

Koutrika and Ioannidis' work [24]: $f_\wedge$ 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 Equation (4.3) and Equation (4.4), respectively.

$$f_\wedge(p_1, p_2) = 1 - (1 - p_1)(1 - p_2) \tag{4.3}$$

$$f_\vee(p_1, p_2) = \frac{p_1 + p_2}{2} \tag{4.4}$$

$f_\wedge$ behaves inflationary whereas $f_\vee$ has a reserved behavior. 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.

During preference selection, we order the preferences by intensity value. However, is important to note that the value returned by the $f_\wedge$ function, when composed, does not change with the order of the preferences.

Proposition 1. *Let P={p_1, p_2, ..., p_n} be a list of n preferences, where by p_1 we denote the intensity of preference P1. When predicates, selected from P, are combined using only AND operator, the combined intensity value computed using $f_\wedge$ function is independent of the order in which preferences are selected and considered.*

Proof. We will prove by induction that $f_\wedge(p_1, p_2 \ldots p_n) = 1 - (1 - p_1) * (1 - p_2) * \ldots * (1 - p_n)$.

Step 1:

Let S_P={p_1, p_2, p_3}, where intensity(p_1) > intensity(p_2) > intensity(p_3).

$f_\wedge(p_1, p_2) = 1 - (1 - p_1) * (1 - p_2)$, by definition.

Case 1: $f_\wedge(p_1, p_2, p_3) = f_\wedge(p_1, f_\wedge(p_2, p_3)) = f_\wedge(p_1, [1 - (1 - p_2) * (1 - p_3)]) =$

$= 1 - (1 - p_1) * [1 - [1 - (1 - p_1) * (1 - p_2)]] = 1 - (1 - p_1) * (1 - p_2) * (1 - p_3)$

Case 2: $f_\wedge(p_1, p_2, p_3) = f_\wedge(p_2, f_\wedge(p_1, p_3)) = f_\wedge(p_2, [1 - (1 - p_1) * (1 - p_3)]) =$

$= 1 - (1 - p_2) * [1 - [1 - (1 - p_1) * (1 - p_3)]] = 1 - (1 - p_1) * (1 - p_2) * (1 - p_3)$

Case 3: $f_\wedge(p_1, p_2, p_3) = f_\wedge(p_3, f_\wedge(p_1, p_2)) = f_\wedge(p_3, [1-(1-p_1)*(1-p_2)]) =$
$= 1-(1-p_3)*[1-[1-(1-p_1)*(1-p_2)]] = 1-(1-p_1)*(1-p_2)*(1-p_3)$

Induction step:

We assume that $f_\wedge(p_1, p_2, \ldots, p_n) = 1-(1-p_1)*(1-p_2)*\ldots*(1-p_n)$.

We want to prove that $f_\wedge(p_1, p_2, \ldots, p_{n+1}) = 1-(1-p_1)*(1-p_2)*\ldots*(1-p_{n+1})$.

$f_\wedge(p_1, p_2, \ldots, p_{n+1}) = f_\wedge(p_1, p_2, \ldots, p_n, p_{n+1}) = f_\wedge(p_{n+1}, f_\wedge(p_1, p_2, \ldots, p_n)) =$
$= 1-(1-p_{n+1})*(1-f_\wedge(p_1, p_2, \ldots, p_n))$.

From the induction step we know that:

$f_\wedge(p_1, p_2, \ldots, p_n) = 1-(1-p_1)*(1-p_2)*\ldots*(1-p_n)$
$\Rightarrow f_\wedge(p_1, p_2, \ldots, p_{n+1}) = 1-(1-p_{n+1})*[1-(1-(1-p_1)*(1-p_2)*\ldots*(1-p_n))] =$
$= 1-(1-p_{n+1})*(1-p_1)*(1-p_2)*\ldots*(1-p_n) = 1-(1-p_1)*(1-p_2)*\ldots*(1-p_{n+1})$.

Therefore, the order in which we apply the $f_\wedge$ function does not change the final result. $\square$

Proposition 2. *Let P=$\{P_1, P_2, P_3\}$ be a list of three preferences, where p_i is the intensity value of preference P_i. When these preferences are combined using only OR operator, the combined intensity value computed using $f_\vee$ function depends on the order the preferences are selected. Moreover, we have: $f_\vee(p_1, f_\vee(p_2, p_3)) \geq f_\vee(p_2, f_\vee(p_1, p_3)) \geq f_\vee(p_3, f_\vee(p_1, p_2))$.*

Proof. Let us assume that the $p_1 \geq p_2 \geq p_3$. This assumption does not influence the proof since if there is a different order we can easily order preferences descending by their intensity value and reassign the name of the variables representing the intensity value for each preference.

We also have:

$$f_\vee(p_1, f_\vee(p_2, p_3)) = \frac{p_1 + f_\vee(p_2, p_3)}{2} = \frac{p_1 + \frac{p_2+p_3}{2}}{2} = \frac{2*p_1+p_2+p_3}{4}$$
$$f_\vee(p_2, f_\vee(p_1, p_3)) = \frac{p_2 + f_\vee(p_1, p_3)}{2} = \frac{p_2 + \dfrac{p_1+p_3}{2}}{2} = \frac{2*p_2+p_1+p_3}{4}$$
$$f_\vee(p_3, f_\vee(p_1, p_2)) = \frac{p_3 + f_\vee(p_1, p_2)}{2} = \frac{p_3 + \dfrac{p_1+p_2}{2}}{2} = \frac{2*p_3+p_1+p_2}{4}$$

And since $p_1 \geq p_2 \geq p3 \Rightarrow f_\vee(p_1, f_\vee(p_2, p_3)) \geq f_\vee(p_2, f_\vee(p_1, p_3)) \geq f_\vee(p_3, f_\vee(p_1, p_2))$

$\square$

To show how the intensity value influences the final result, we bring back the canonical example given in the Background section and we add the intensity values for each preference.

Example 6. *Assume the three preferences over car entities:*

$P1$: *"I like a car with the price between \$7,000 and \$16,000 with intensity 0.8".*

$P2$: *"I prefer car with a mileage between 20,000 and 50,000 with intensity 0.5".*

$P3$: *"I prefer a* BMW *or a* Honda *with intensity 0.2".*

A snapshot of the dealership relation is given in the Table 8

Table 8: Dealership Relation

id	**price**	**mileage**	**make**
t1	\$7,000	43,489	Honda
t2	\$16,000	35,334	VW
t3	\$20,000	49,119	Honda

Since all preferences are defined on different attributes, using the intensity values and the combination function given in Equation (4.3) we can now compute a combined intensity value for all the tuples.

Tuple t1 matches all three preferences. Therefore, the combined intensity value is:

$$f_\wedge(f_\wedge(P1, P2), P3) = f_\wedge(f_\wedge(0.8, 0.5), 0.2) = f_\wedge(0.9, 0.2) = 0.92$$

Tuple t2 matches only first and second preference. The combined intensity value is:

$$f_\wedge(P1, P2) = f_\wedge(0.8, 0.5) = 0.9.$$

Tuple t3 matches only the last two preferences. The combined intensity value is:

$$f_\wedge(P2, P3) = f_\wedge(0.5, 0.2) = 0.6.$$

The final results are given in Table 9.

Table 9: Intensities Values for Tuples in Dealership Relation

t1	0.92	tuple matches all three preferences
t2	0.9	tuple matches preference P1 and P2
t3	0.6	tuple matches preference P2 and P3

4.7 SUMMARY

In this section we described the necessary steps to transition from the theoretical model to a real system implemented in Neo4j. Although multiple options exist to store a graph, we picked the graph database implementation to overcome the limitations provided by the other options in terms of scalability – for the adjacency matrix model – and graph traversal – for the relational database model.

Moreover, we showed how we can convert qualitative preferences into quantitative preferences using one of the two functions presented in Equation (4.1) and Equation (4.2), process which allows us to "cover", with the existing preferences, more tuples from the dataset.

Finally, we showed how intensity value changes when the preferences are combined and we discussed the reasoning behind selection an inflationary function – when preferences are combined with an AND operator – and a reserved function, when preferences are combined with an OR operator.

5.0 TOWARDS A PRACTICAL AND EFFICIENT ALGORITHM FOR GENERATING BEST PREFERENCE COMBINATIONS

Given a set of preferences, each with an intensity value, we want to be able to determine the best combination of preferences that, in one hand, maximizes the combined intensity value and, in the other hand, still returns tuples. In that respect, we expect the intensity and the number of returned results to play an important role in determining the best strategy for combining preferences.

In this chapter we describe an efficient algorithm for generating the best combination of preferences. In this context, and in the rest of this dissertation, we use, in many occasions, the term applicable combination.

Definition 15. *Applicable combination – Given a base select query, a preference enhanced query is created by adding in the WHERE clause any predicate or combinations of predicates from the user's profile. Any such predicate combination that returns at least one tuple when is used to enhance a base select query is called an applicable combination.*

5.1 UTILITY AND COVERAGE METRICS

In the next sections we are introducing two new metrics used to evaluate the performance of different algorithms designed to combine preferences in our proposed framework.

5.1.1 Utility Metric

In Section 4.6.1 we define the preference composition function, when preferences are combined with an AND operator, as a function with an inflationary behavior, given by Equation (4.3). Because of the way it is defined, we expect that combining preferences will give us a better intensity value than using one preference at a time. However, it is also important to see how the number of tuples returned varies and we cannot expect that the intensity and the number of tuples are correlated (as we also confirm from our experiments). As such, it is important to come up with a metric that combines both.

First, we need to record the number of tuples returned by using one particular preference or one combination of preferences.

Definition 16. *Preference Selectivity – The ratio between the total number of tuples returned and the number of predicates used to enhance a base query. This predicate can be an atomic one (i.e., only one predicate used) or it can be a combination of multiple preference predicates.*

$$\mathit{Pref\_Selectivity} = \frac{\#tuples}{\#preferences} \tag{5.1}$$

Definition 17. *Utility – A metric defied as a product between the preference selectivity and the combined intensity value.*

$$\mathit{Utility} = \mathit{Pref\_Selectivity} * intensity \tag{5.2}$$

5.1.2 Coverage Metric

Definition 18. *Coverage – The total possible number of tuples "touched" when all preferences are used independently.*

We use *coverage* metric to record how many tuples a user can access using his/hers preferences. When preferences are very selective, a system might want to combine preferences using an OR semantics in order to bring as many tuples as possible into the result list.

Coverage is a useful metric that describes how selective a set of preferences is. However, there is no obvious way to decide what is more important: a preference that returns millions of tuples or a preference that returns only few. Moreover, if the intensity value attached to the tuples returned is very small, then it seems reasonable to pick the preferences that return less tuples first, but with higher intensity value.

5.2 THEORETICAL UPPER BOUND COMPLEXITY FOR PREFERENCE COMBINATION

In previous chapter, Section 4.6, we discussed about three different ways to combine preferences: conjunctive, disjunctive, and mixed clause.

In principle, instead of deciding online what is the best combination that returns a non empty result, with the highest intensity value, we can first compute the combined intensity values for all possible combinations and then, enhance the MySql query with one preference at a time until enough results are returned. However, Proposition 3 and Proposition 4 show that the number of combinations that a system needs to produce increase exponentially in number of preferences, therefore is not a viable solution.

Proposition 3. *Given a list of N preferences, the theoretical upper bound of the preference combination problem, using only the AND operator, is* $2^N - 1$.

Proof. Assume we have the following list of preferences: P $=\{p_1, p_2, p_3, p_4, p_5\}$.

In the trivial case, there are $\binom{N}{1} = N$ possible combinations of one preference, which is, P, the initial list of preferences.

When combining two preferences, out of all N, we have $\binom{N}{2}$ possible ways to choose 2 preferences out of all N. Each preference is combined once with one of the preferences that succeed it in the list of ordered preferences. Since we only combine preferences with an AND operator, for the second step we have $\binom{N}{2}$ number of combinations.

When combining three preferences, out of N, we have $\binom{N}{3}$ possible combinations.

...

Following the same counting strategy, for all the remaining steps, we infer that the number of all possible combinations is:

$$S = \binom{N}{1} + \binom{N}{2} + \binom{N}{3} + \ldots + \binom{N}{N} = \sum_{k=0}^{N} \binom{N}{k} - \binom{N}{0} = 2^N - 1. \tag{5.3}$$

□

Proposition 4. *Given a list of N preferences, the theoretical upper bound of the preference combination problem, using both AND and OR operators, is* $\dfrac{3^N - 1}{2}$.

Proof. As before, assume we have the following list of preferences: P $=\{p_1, p_2, p_3, p_4, p_5\}$.

In the trivial case, there are $\binom{N}{1}$ = N possible combinations of one preference, which is, P, the initial list of preferences.

When combining two preferences, out of all N, we have $\binom{N}{2}$ possible combinations. Each preference is combined once with one of the preferences that succeed it in the list of ordered preferences. However, since each combination can be created using an AND or an OR operator, there are twice as many possible combinations making a total of $2\times\binom{N}{2}$ possible combinations. In our example, the resulted list is P2 = P2a ∪ P2b where :

- P2a $=\{(p_1$ AND $p_2)$, $(p_1$ AND $p_3)$, $(p_1$ AND $p_4)$, $(p_1$ AND $p_5)\}$
- P2b $=\{(p_1$ OR $p_2)$, $(p_1$ OR $p_3)$, $(p_1$ OR $p_4)$, $(p_1$ OR $p_5)\}$

When combining three preferences, out of all N, we have $\binom{N}{3}$ possible combinations. Same as before, because we can combine the third preferences with an AND or an OR, there are twice as many preference combination possible. Moreover, the previous combinations of two can be again made using an AND or an OR therefore there are in total $2^2\times\binom{N}{3}$ possible combinations. In our example, the resulted list of all possible combinations of three preferences is P3 =P3a ∪ P3b ∪ P3c ∪ P3d where:

- P3a $=\{(p_1$ AND p_2 AND $p_3)$, $(p_1$ AND p_3 AND $p_4)$, $(p_1$ AND p_4 AND $p_5)$ $\}$
- P3b $=\{(p_1$ AND p_2 OR $p_3)$, $(p_1$ AND p_3 OR $p_4)$, $(p_1$ AND p_4 OR $p_5)$ $\}$
- P3c $=\{(p_1$ OR p_2 AND $p_3)$, $(p_1$ OR p_3 AND $p_4)$, $(p_1$ OR p_4 AND $p_5)$ $\}$
- P3d $=\{(p_1$ OR p_2 OR $p_3)$, $(p_1$ OR p_3 OR $p_4)$, $(p_1$ OR p_4 OR $p_5)$ $\}$

Following the same reasoning, we have $2^{N-1}\binom{N}{N}$ combinations when all the preferences are combined.

The total number of combination is given by the sum S in Equation (5.4).

$$S = \binom{N}{1} + 2^1 \times \binom{N}{2} + 2^2 \times \binom{N}{3} + \ldots + 2^{N-1} \times \binom{N}{N} \tag{5.4}$$

We also have that Equation (5.5) holds for every x.

$$\sum_{k=0}^{N} x^k \times \binom{N}{k} = (x+1)^N \tag{5.5}$$

In order to use Equation (5.5), we need to add to S the missing binomial term, $\binom{N}{0}$=1. In this case, using Equation (5.4) and Equation (5.5) we have that:

$$2 \times S + \binom{N}{0} = (2+1)^N \Rightarrow S = \frac{3^N - 1}{2}. \tag{5.6}$$

□

To eliminate the exponential complexity we need to be able to decide, beforehand, which combinations are applicable and which combinations return a better combined intensity value. This will allow us to reduce the space of combinations that should be executed at the running time. In the next sections we show how we can identify a pruning mechanism to alleviate the time complexity problem.

5.3 ALGORITHMS TO GENERATE PREFERENCE COMBINATIONS

To show what happens with the intensity value when we combine preferences, we designed three algorithms that combine a list of preferences. For all algorithms:

- Input: a list of preferences, for one particular user, in descending order of intensity value
- Output: a list L that records, for each preference combination created:
  <number of predicates used, number of tuples returned, combined intensity value>

The first algorithm – *Combine-Two* – combines only *two* preferences and its design and behavior is explained in Section 5.3.1. The second algorithm – *Partially-Combine-All* – combines all available preferences, one at a time, using a mixed clause semantic and outputs the number of preferences used, the number of tuples returned and the combined intensity value. More details about this algorithm are presented in Section 5.3.2. Finally, the last algorithm designed to show the difficulties in determining the best combination – *Bias-Random-Selection* – combines all available preference and decides, using a biased coin flip, if the preference should be included or not. More details about this algorithm are presented in Section 5.4.

5.3.1 Combine-Two Algorithm

The *Combine-Two* algorithm is an exhaustive combination of two preferences, with one preference kept fix, and a different preference selected at every step from the remaining list. This algorithm takes as input a list of preferences ordered descending by intensity value and returns a list of preference combinations along with theirs combined intensity value and the number of tuples returned.

The *Combine-Two* Algorithm makes combinations that contain only two preferences. At every step, the current preference is combined with all the remaining preferences in the input list, one at a time.

In Section 4.6 we defined the *conjunctive, disjunctive* and *mixed* clause when we discussed about the three obvious ways to combine predicates. In this chapter we are using the following notations:

- AND semantics: creates a conjunctive clause and all predicates are connected using only an AND operator
- OR semantics: creates a disjunctive clause and all predicates are connected using only an OR operator
- AND_OR semantics: creates a mixed clause and all predicates that refer to the same attribute are combined using an OR operator, whereas all predicates that contain different attributes are combined using an AND operator

Algorithm description. The Combine-Two algorithm starts with the preference with the highest intensity value and appends next available preference from the remaining list, using AND_OR semantics, Algorithm 2 or AND semantics, Algorithm 3 . Then it continues with the second most preferred and one preference from the remaining list of preferences.

In the case of AND semantics, some of the combinations will not return any tuples (e.g., venue='SIGMOD' and venue='VLDB'). To eliminate the cases where no tuples are returned, the AND_OR semantics combine two preferences on the *author* attribute with OR semantics and two preferences, one preferences on *author* attribute and the other on the *venue*, with AND semantics.

In Algortithm 3, AND () function takes as arguments two preferences, given as a pair <predicate, intensity value> and returns a combined predicate using an AND operator, with the combined intensity value computed using Equation (4.3) attached.

In Algortithm 2, AND () function behaves in the same ways as explained before. The OR () function takes, again as arguments, two preferences given as a pair <predicate, intensity value> and returns a combined predicate using an OR operator, with the combined intensity value computed using Equation (4.4) attached. The OR () function is applied when the two predicates refer to the same attribute, whereas the AND () function applies when the two predicates refer to different attributes (e.g., one predicate is on the *venue* and the other one is on the *author*).

The *runQuery ()* function is called after a predicate combination is created (i.e, predicates are combined using AND or OR operators and the combined intensity value is computed). This function perform three tasks:

1. It enhances the base_query with the predicate stored in the preference argument
2. It runs the new query over the database tuples
3. Returns the results as: <2, #tuples, combine intensity value>. The combined intensity value is previously computed and stored in the preference argument of this function

Example 7. *Let P ={ P_1, P_2, P_3} be a list of preferences for one particular user, order descending by intensity value. Also, we assume the following predicates for each preference:*

- P_1: { *predicate: "dblp.venue = INFOCOM"*}
- P_2: { *predicate: "dblp_author.aid = 2222"*}
- P_3: { *predicate: "dblp_author.aid = 4787"*}

With the preferences listed in the Example 7, on one hand, Combine-Two algorithm with AND_OR semantics will run queries like:

```
SELECT count(distinct dblp.pid)
FROM dblp join dblp_author on dblp.pid =dblp_author.pid
WHERE dblp.venue="INFOCOM'' AND dblp_author.aid=2222;


SELECT count(distinct dblp.pid)
FROM dblp join dblp_author on dblp.pid =dblp_author.pid
WHERE dblp.venue="INFOCOM'' AND dblp_author.aid=4787;


SELECT count(distinct dblp.pid)
```

Algorithm 2 Combine-Two Algorithm with AND_OR sematics

Input: *allPref* – list of preferences, for one particular user, ordered descending by intensity value

Output: L={ p, p=<2, #tuples, combined intensity value>}.

```
1:  BEGIN
2:  L ← ∅;
3:  base_query = “SELECT count(distinct dblp.pid)” +
4:  + “FROM dblp join dblp_author on dblp.pid=dblp_author.aid ” +
5:  + “WHERE ”
6:  while (there are more preferences left) do
7:     p1 = allPref.readNextAvailablePreference();
8:     nextPrefList ← allPref - p1;
9:     while (there are more preferences left in nextPrefList) do
10:       p2 = nextPrefList.readNextAvailablePreference();
11:       if (p1 and p2 have the same attribute) then
12:          preference = OR (p1, p2);
13:       else
14:          preference = AND (p1, p2);
15:       end if
16:       L ← =runQuery (base_query, preference)
17:    end while
18: end while
19: RETURN L;
20: END
```

Algorithm 3 Combine-Two Algorithm with AND sematics

Input: *allPref* – list of preferences, for one particular user, ordered descending by intensity value

Output: L={p, p=<2, #tuples, combined intensity value>}

```
1:  BEGIN
2:  L ← ∅;
3:  base_query = “SELECT count(distinct dblp.pid)” +
4:  + “FROM dblp join dblp_author on dblp.pid=dblp_author.aid ” +
5:  + “WHERE ”
6:  while (there are more preferences left) do
7:      p1 = allPref.readNextAvailablePreference();
8:      nextPrefList ← allPref - p1;
9:      while (there are more preferences left in nextPrefList) do
10:         p2 = nextPrefList.readNextAvailablePreference();
11:         preference = AND (p1, p2);
12:         L ← =runQuery (base_query, preference)
13:     end while
14: end while
15: RETURN L;
16: END
```

```
FROM dblp join dblp_author on dblp.pid =dblp_author.pid
WHERE dblp_author.aid=2222 OR dblp_author.aid=4787;
```

As explained before, we modified the Algorithm 2 to combine all the preferences only with the AND operator, in Algorithm 3. To do that, we eliminate the extra step that checks if the two preferences refer to the same attribute and the call of OR() function. Other than these modifications the algorithm is the same and, using the preferences given in Example 7, this algorithm will run queries like:

```
SELECT count(distinct dblp.pid)
FROM dblp join dblp_author on dblp.pid =dblp_author.pid
WHERE dblp.venue="INFOCOM'' AND dblp_author.aid=2222;

SELECT count(distinct dblp.pid)
FROM dblp join dblp_author on dblp.pid =dblp_author.pid
WHERE dblp.venue="INFOCOM'' AND dblp_author.aid=4787;

SELECT count(distinct dblp.pid)
FROM dblp join dblp_author on dblp.pid =dblp_author.pid
WHERE dblp_author.aid=2222 AND dblp_author.aid=4787;
```

Complexity. This algorithm computes all combinations of two in $O(N^2)$, where N is the size of the preference list given as input, because there are $\binom{N}{2} = \frac{N!}{(N-2)!*2!} = \frac{(N-1)*N}{2}$.

5.3.2 Partially-Combine-All Algorithm

The *Partially-Combine-Two* algorithm takes as input a list of preferences ordered descending by intensity value. As before, the algorithm returns a list L of pairs: L={ p, p=<#predicates, #tuples, combined intensity value>}. The purpose of this algorithm is to combine as many preferences as possible, since this process will return the most preferred tuples, customized based on a user profile. However, some combination as not applicable and to avoid creating combinations that do not return anything we combine all predicates that refer to the same attribute using an OR operator and all predicates that refer to different attributes using an AND operator. Moreover, because any combination with an AND operator has an inflationary behavior (i.e., the combined intensity value is greater that any intensity value participating in the combination) we want to run as many queries with AND as possible.

Algorithm description.The Partially-Combine-All algorithm, presented in Algorithm 4, starts with an empty preference predicate and appends, one at a time, a preference retrieved from the ordered list. For each predicate retrieved, this algorithm stores in *attributesUsed* list any attribute that was not used before. Moreover, the algorithm stores in *queriesRan* all predicates combination made so far for future references.

When predicates are combined, there are three possible conditions to be verified in order to decide how should the predicates be combined.

- Condition 1: if the new predicate contains an attribute that was not used before in any combinations, we rerun all the queries we ran before, appending the new preference to the query predicate, using an AND operator. The idea behind this behavior comes from the fact that we want to create as many combinations as possible using AND operator.
- Condition 2: If the new preference contains an attribute that was already used, but the query contains only one attribute type, we append the new predicate to the last query created, using OR. In this case, because the combined intensity value will decrease, we only want to append the new predicate to the last combination.
- Condition 3: If the new preference contains a predicate that was already used, and there are at least two different attributes used in the last combination of preferences (i.e., there is an AND between two preferences) then:
  - the algorithm runs all the queries ran before, that do not contain the new attribute, inserting the new predicate with an AND operator (i.e., AND(queriesRan, pi) function call)
  - the algorithm adds the new predicate into the appropriate clause of the last combination, using an OR operator.

Algorithm 4 Partially-Combine-All Algorithm

Input: *allPref* -a list of all preferences for one particular user

Output: L = {p, p=<#predicates, #tuples, combined intensity value>}.

```
1: BEGIN
2: L ← ∅ ; query ← ∅ ; attributesUsed ← ∅ ; queriesToRan ← ∅ ; queriesRan ← ∅ ;
3: base_query = "SELECT count(distinct dblp.pid)" +
4: + "FROM dblp join dblp_author on dblp.pid=dblp_author.aid WHERE "
5: while (there are more preferences in allPref list) do
6:    pi =allPref→read_and_remove_a_pref
7:    if (query is empty) then
8:       query =base_query + pi
9:       attributesUsed ← pi.attribute
10:   else if ( pi.attribute ∉ attributesUsed) then
11:      for (all combinations already created) do
12:         queriesToRan ← AND(queriesRan, pi)
13:      end for
14:      attributesUsed ← pi.attribute
15:   else
16:      lastCombination ← queriesRan→last
17:      if ( pi.attribute ∈ attributesUsed and lastCombination does NOT contain AND) then
18:         queriesToRan ← OR(lastCombination, pi)
19:      else if ( pi.attribute ∈ attributesUsed and lastCombination contains AND) then
20:         queriesToRan ← OR(queriesRan, pi)
21:         queriesToRan ← AND(queriesRan, pi)
22:      end if
23:   end if
24:   L ← runQuery(queriesToRun, queriesRan);
25:   queriesToRun ← ∅
26: end while
27: RETURN L
28: END
```

Using again the list of preferences given in Example 7, the *Partially-Combine-All* algorithm creates the following predicates combinations:

Start: *empty string*

Combination 1: dblp.venue=”INFOCOM”

Combination 2: dblp.venue=”INFOCOM” AND dblp_author.aid=2222

Combination 3: dblp.venue=”INFOCOM” AND dblp_author.aid=4787

Combination 4: dblp.venue=”INFOCOM” AND (dblp_author.aid=4787 OR dblp_author.aid=2222)

Complexity.

Proposition 5. *Complexity of Partially-Combine-All algorithm – Given a list of N preferences, in the best case, the Partially-Combine-All algorithm runs in O(N) time, when no previously made combinations are used. However, when the algorithm uses previous combinations, the running time is O(N^2).*

Proof. Best Case

[1] In the case when all preferences defined in a user profile contain only one attribute, the algorithm appends to the last combination the new predicate using an OR operator. In this case, the algorithm runs N queries, therefore there is a O(N) running time.

[2] In the case when all preferences contain one specific attribute except for the first one, the algorithm still runs in O(N). Let us assume P=$\{$v, a_1, a_2, . . ., $a_{N-1}\}$ be the list of predicates where v contains one attribute which is different than the attribute contained in $a_i \forall i <= N-1$. In this case, we will have the following combinations:

1. for v, creates combination: v
2. for a_1 creates combination: v AND a_1
3. for a_2 creates combination:
    - v AND a_2
    - v AND (a_1 OR a_2)
4. for a_3 creates combination:
    - v AND a_3
    - v AND (a_1 OR a_2 OR a_3)

5. ...
6. for a_{N-1} creates combination:
   - v AND a_{N-1}
   - v AND (a_1 OR a_2 OR a_3 OR ... a_{N-1})

In this case, the algorithm runs one query for first and second predicate and two queries for all the remaining predicates. This creates 2*N-2 operations which will be executed in O(N) time.

[3] In the case when all preferences contain one specific attribute except for the last one, the algorithm still runs in O(N). Let us assume P=$\{a_1$, a_2, ..., a_{N-1}, v$\}$ be the list of predicates where v contains one attribute which is different than the attribute contained in a_i $\forall\ i \leq N-1$. In this case, we will have the following combinations:

1. for a_1 creates combination: a_1
2. for a_2 creates combination: a_1 OR a_2
3. for a_3 creates combination: a_1 OR a_2 OR a_3
4. ...
5. for a_{N-1} creates combination: a_1 OR a_2 OR ... a_{N_1}
6. for v creates combinations:
   - a_1 AND v
   - a_1 OR a_2 AND v
   - ...
   - a_1 OR a_2 OR ... a_{N_1} AND v

In this case, the algorithm runs N-1 steps for all a_i preferences and another N-1 steps for preference v $\Rightarrow$ O(N) running time.

<u>Worst Case</u>

In the case when the list of preferences contain two different predicates, the worst case comes when there are $\frac{N}{2}$ predicates of one type and $\frac{N}{2}$ predicates of the other type. In this case, the algorithm combines the first $\frac{N}{2}$ predicates that have the same attribute using and OR operator which is executed in $\frac{N}{2}$ steps. Then, for the remaining $\frac{N}{2}$ preferences, all with the same attribute but different that the first attribute, the algorithm creates $\frac{N}{2}$ combinations using AND operator plus another $\frac{N}{2}$ combinations using OR operator. In this case, the running time is $\frac{N}{2}+\frac{N}{2}*(\frac{N}{2}+\frac{N}{2})$ $\Rightarrow O(N^2)$ $\square$

5.4 BIAS-RANDOM-SELECTION ALGORITHM

The last algorithm randomly selects a predicate to be included in the predicate combination. Bias-Random-Selection algorithm, Algorithm 5, decides with a biased coin flip if a given preference should be included or not in the generated preference combination. We bias the coin flip towards the preferences with higher intensity values, since this will give us a better combined intensity value for the returned tuples.

Algorithm description. The algorithm takes as input a list of all preferences for one particular user, ordered descending by intensity values, The output of this algorithm is represented by a list that contains records of the form: <#predicates combined, #tuples returned, combined intensity value>.

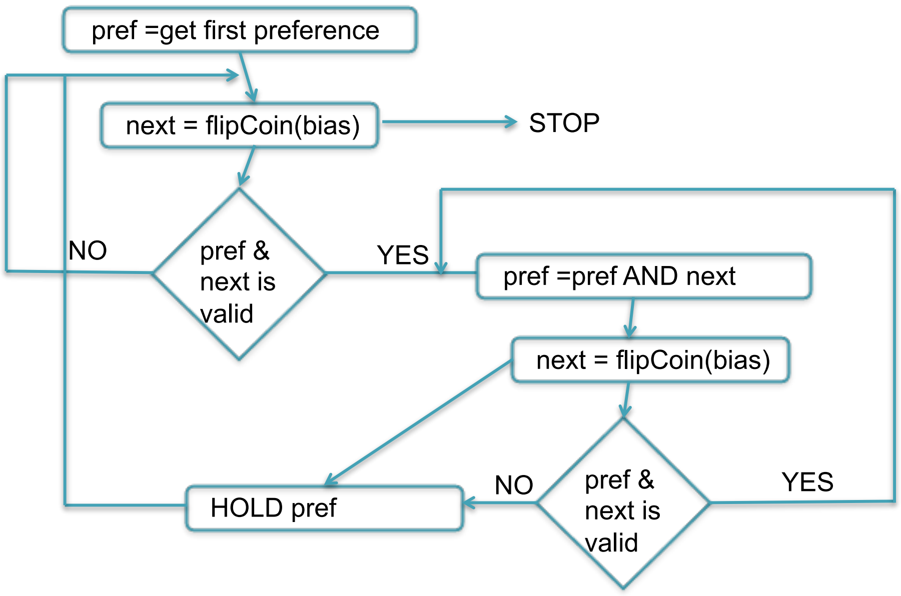


Figure 16: Bias-Random Algorithm Representation

The main part of the algorithm is described in Figure 16. These steps are executed for all the preferences stored in a user profile and every time the subroutine is execute, the initialization "pref=get first preference" will select the next available preference in the list. The pseudocode of Bias-Random-Selection algorithm is given in Algorithm 5.

The predicate combination subroutine has six steps:

- Step 1: With the fist preference already selected, the algorithm looks in the list of remaining preferences, and it decides to keep or to skip a preference based on a biased coin flip. When one preference is selected, it is returned to the subroutine.
- Step 2: The first combination is created by connecting the first preference and the preference selected in Step 1 with an AND operator. If this combination is applicable (i.e., it returns tuples), then the algorithm temporary stores this combination as the current preference.
- Step 3: From the remaining list of preferences (i.e., all preferences that follow the last preference selected) the algorithm picks another randomly selected preference and check the applicability of the new combination.
- Step 4: If the new combination is not applicable, then the algorithm runs the previous combination and adds to the output list the results (i.e., number of predicates in the combination, number of tuples returned and the combined intensity value) and executes again Step 1.
- Step 5: If the new combination is applicable, the algorithm creates a new temporary predicate combination and returns to Step 3.
- Step 6: If the flipCoin() function does not return any preference because there are no more preferences left, the algorithm behaves as in Step 4.
- Step 7: The algorithm stops when there are no more preferences that can be selected.

Complexity. The complexity of this algorithm depends on the number of preferences selected using the biased coin flip. In the worst case, when all preferences are selected, and all combinations are applicable, the algorithm creates, for each preference, K^2 combinations, where K is the size of the remaining list of preferences. For example, for the third preference, there are N-3 preferences left that can be combined with the third preference. Because of this, the running time of this algorithm is $O(N^3)$.

Algorithm 5 Bias-Random-Selection

Input: allPref – a list of all preferences for one particular user

Output: orderedL = {p, p=<#predicates, #tuples, combined intensity value >}

```
1:  BEGIN
2:  oderedL ← ∅ ; nextL ← ∅ ;
3:  for ( all preference ids ) do
4:    first = allPref → preference_id;
5:    while ( there are more preferences left ) do
6:      preference_id =flipCoin(bias)
7:      second = allPref → preference_id;
8:      if ( “first AND second” does not return any tuple ) then
9:        continue; {/* first combination is not applicable. Try a new combination */}
10:     else
11:       pref = “first AND second”;
12:     end if
13:     while ( there are preferences left ) do
14:       next_preference_id =flipCoin(bias)
15:       next =allPref → next_preference_id;
16:       if ( “pref AND next” does not return any tuple ) then
17:         orderedL ← runQuery(pref)
18:         break; {/* exit the inner loop. No more applicable combinations can be created */}
19:       else
20:         pref = “pref AND next”
21:       end if
22:     end while
23:   end while
24: end for
25: END
```

5.5 AN ALGORITHM FOR A PRACTICAL AND EFFICIENT PREFERENCE SELECTION

All previous algorithms demonstrate the difficulties related to preference combinations. Our *Practical and Efficient Preference Selection* (PEPS) algorithm is created to overcome these difficulties and return a sorted list of preferences based on the combined intensity value.

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.

To create applicable combinations of predicates we have implemented a first and complete version of the PEPS algorithm and an approximation version for the same algorithm. Both versions make use of a pre-computed list of combinations of two predicates, which is updated when the preference graph is updated. Each item in this list contains the pair of predicates that are AND combined, the pre-computed combined intensity value, and a count of number of tuples returned when the predicate combination is used. The PEPS algorithms use this list to retrieve all the valid combinations that start with a particular predicate.

5.5.1 The Complete PEPS Algorithm

The Complete PEPS algorithm, Algorithm 6 uses AND semantics to combine as many predicates as possible.

The algorithm iterates over the list of preferences and for a given preference *p*, selects all the items from list of combinations of two predicates, with the combined intensity value greater than the intensity value of *p*. Next, it also selects all other combinations, that do not have the combined intensity value greater than the intensity value of *p*, but given enough extra predicates, the final value can still be greater than *p*. We based our algorithm on Proposition 6. 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.

Proposition 6. *Let P = { P_1, P_2, P_3, . . ., P_n } be a list of preferences, ordered descending by their intensity value, and intensity(P) = { p_1, p_2, p_1, . . ., p_n } be their associated intensity values. If the combined intensity value of P_2 and P_3 is not greater than p_1 then it can only be greater if there are at least K = $\frac{log(1-p1)}{log(1-p2)}$ more preferences in the list, with intensity value equal to p_2.*

Proof. The combined intensity value for P_2 and P_3 is given by:

$f_\wedge(p_2, p_3) = 1 - (1 - p_2) * (1 - p_3)$

But since $p_2 \geq p_3 \Rightarrow f_\wedge(p_2, p_3) \leq 1 - (1 - p_2)^2 \Rightarrow f_\wedge(p_2, p_3) \leq 1 - (1 - p_2)^K$

We are looking for a K value such that $f_\wedge(p_2, p_3) \geq p_1$

$f_\wedge(p_2, p_3) \geq p_1 \Rightarrow 1 - (1 - p_2)^K \geq p_1 \Rightarrow 1 - (1 - p_2)^K \geq 1 - (1 - p_1) \Rightarrow (1 - p_2)^K \leq (1 - p_1) \Rightarrow K \geq \frac{log(1-p1)}{log(1-p2)}$ □

The previous proposition is an optimistic approximation of the number of preferences that need to be combined in order to have a combination with intensity value greater than p_1. This is because it assumes that all preferences, except the first one, have the intensity value equal to p_2. However, the intensity value might decrease for each preference that follow p_2 in the list of preferences. This proposition gives us a lower bound on the number of predicates that need to be combined in order to have a combination with an intensity value as good as the intensity value of p_1.

Algorithm description. Complete PEPS uses two stacks. First stack, *CombStack* is initialize with all the combinations of two predicates found in a previous step or all combinations of two preferences with combined intensity value greater than the intensity value of *p* for the first time (i.e., line 2 in Algorithm 6) or with possibility of being greater if enough predicates are combined. 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* (i.e., 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 applicability of the new combination by verifying that there is an applicable combination between all predicates already used and p_j. If the new combination is applicable, 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 in the ORDER list. 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 applicability is checked again, as explained before. The algorithm continues until there are no more combinations left in *CombStack*.

Complexity. The complexity of this algorithm depends on the number of applicable pairs of predicates combinations. In the worst case, this algorithm behaves as the exhaustive search, as stated in Proposition 3.

5.5.2 The Approximate PEPS Algorithm

The second version of PEPS Algorithm, Approximate PEPS, differs from the Complete PEPS algorithm only at the first step when the combinations of two preferences are selected. The Complete PEPS algorithm keeps in the working set all combinations that might be useful later, to make sure no possible combination is lost. The Approximate PEPS algorithm removes this requirement for a faster tuple retrieval. The algorithm store in the working set only these combinations of predicates that have the combined intensity value greater than p_1. The Algorithm 6 is still applicable for the Approximate PEPS algorithm because the list of combinations of two preferences, the starting point of the algorithm, is pre-computed before this routine is called.

The benefit of the Approximate PEPS Algorithm comes from the fact that the algorithm is cutting out combinations that are probably not useful (i.e., they will not create an applicable predicate combination with the combined intensity value as large as required). However, this algorithm is only an approximate Top-K algorithm because it is possible to miss tuples that would have been returned by one of the combinations pruned.

Algorithm 6 PEPS Algorithm with AND semantics

Input:

CombsOfTwo -a list of valid combinations ordered by the combined intensity value

p -the next preference with highest intensity value

Output: *ORDER* -a list of predicate combinations ordered by the combined intensity value

```
1: BEGIN
2: CombStack ← CombStack ∪ CombsOfTwo(p) ; PrefStack ← ∅
3: while (CombStack NOT ∅) do
4:    ($p_i$, $p_j$) =CombStack.removeFirst()
5:    lastPref =PrefStack.readFirst()
6:    if (lastPref does NOT exist) then
7:       newPref =AND($p_i$,$p_j$)
8:    else
9:       if (lastPref AND $p_j$ NOT applicable and CombStack does NOT contain ($p_i$, Y) ) then
10:         ORDER ← lastPref ; PrefStack ← remove_last
11:         CombStack.removeAll($p_i$)
12:         go to LINE 4
13:      end if
14:      newPref =AND(lastPref, $p_i$)
15:   end if
16:   nextApplicableCombs ← CombsOfTwo($p_j$)
17:   if (nextApplicableCombs is ∅) then
18:      ORDER ← newPref
19:   else
20:      CombStack ← CombStack ∪ nextApplicableCombs
21:      PrefStack ← PrefStack ∪ newPref;
22:   end if
23: end while
24: RETURN ORDER
25: END
```

5.6 SUMMARY

In this chapter we presented three different algorithms that can be used to combine preferences, each of them with their advantages and disadvantages. The Combine-Two algorithm is a very intuitive one but it limits the combinations to two predicates at a time. It was implemented in two different versions, one with conjunctive clauses only and the other version with mixed clauses. The Partially-Combine-All algorithm uses mixed clauses and creates as many combinations as possible with conjunctive clauses. Bias-Random-Selection algorithm is designed to study the behavior of predicate combination if predicates are selected randomly, where the bias is given by the intensity value of each preference – higher the intensity, higher the chances of the predicate to be selected. Finally, we closed this chapter with our Top-K efficient and practical algorithm, PEPS algorithm.

Moreover, we discussed about the *utility* and *coverage* metric used to characterize the impact of the preferences over the final result.

6.0 EXPERIMENTAL WORKLOAD

To experimentally and fully evaluate our system, we need a large workload consisting of data and associated user preferences, and we need to have both types of preferences defined – qualitative and quantitative preferences. Since such a workload is not available, to the best of our knowledge, we create one by utilizing the DBLP citation dataset and we extract user preferences from the data itself. We describe this process in the next sections.

6.1 DBLP CITATION NETWORK

The DBLP Citation Network V4 dataset [44] 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 and paper id). By parsing this dataset, we create a relational database with four tables:

- *Dblp (pid, title, venue, year, abstract)* – contains basic information about one paper (title, year published, venue) along with a paper id (pid)
- *Author (aid, full_name)* – contains the full name extracted from the dataset and an author id (aid) automatically generated at insertion time
- *Citation (pid, cid)* – contains the citations references, extracted from the DBLP dataset, as a pair of paper ids (i.e., pid, cid). The meaning of an entry in this table is that paper with a given id (i.e., pid) cites the paper with another id (i.e., cid)
- *Dblp_Author (pid, aid)* – contains the links between paper ids (i.e., pid) and the author ids (i.e., aid). This table records what authors published what papers

Table 10: Statistics for the DBLP Database

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

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

- *quantitative_pref (pfid, uid, preference, intensity)*
- *qualitative_pref (pfid, uid, leftPref, rightPref, intensity)*

In both of these tables, the attributes *preference, leftPref* and *rightPref* represent the preference stored as an SQL predicate, *uid* represents the author id and *intensity* is the value of intensity towards that particular preference. In the next section we discuss how we have populated these two preference tables whose arity and cardinality are also shown in Table 10 .

6.2 PREFERENCE EXTRACTION

We extract preferences for multiple users. We define a user as an existing author from the dataset and we use author and user interchangeably. However, new users can be created anytime and inserted in the author table, although they might not have any papers published.

To cover all possible types of preferences described in Section 3.3.1 and Section 3.3.2, we designed the following preference extraction queries:

1. Venue Preference (quantitative preference): User's preference, on a *venue*, based on the venues where he/she published in the past. We create this preference because we assume that someone would prefer to read papers that were published in venues where he/she has published in the past.
2. Author Preference (quantitative preference): User's preference for an *author* based on the co-author information. This preference implies that someone would prefer to read papers published by his/hers coauthors.
3. Preference of one author over another (qualitative preference): For one user, we define the preference on author: "Author A is preferred over author B", meaning that some author is more preferred than other author, for that particular user.
4. Preference of one venue over another (qualitative preference): For one user, we define a preference over some venue, when compared with other venue – "Venue X is preferred over venue Y."
5. Negative Venue Preference (quantitative preference): For one user, we define a negative preference towards the venues where he/she did not publish but other authors that were cited by the user did publish.

We describe the preference extraction process and the formulas used to compute intensity values next. The purpose of this extraction step is to generate meaningful preferences that can be used to test our system; however, the preference extraction problem is orthogonal to this work.

6.2.1 Quantitative Preferences

We extract three types of quantitative preferences.

Preference towards a particular venue. 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, because of this long tail behavior, 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 is equivalent with the user's indifference towards that particular set of tuples. Therefore, the system cannot benefit from having quantitative preferences with intensity equal to zero, and creating a user preference workload with them would be a meaningless exercise.

Preference towards a particular author. Given the *Citation* relation, we find all authors that are cited by a given author. For each user/author, we add one preference 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. At the end, we filter out the preferences with intensity lower than 0.1, since a quantitative preference with intensity value equal to zero means the user is indifferent towards that particular tuple or set of tuples and therefore the preference does not contribute any meaningful information.

Negative 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[1]. On the other hand, if author B published a lot in a venue where author A did not publish, but author A is almost indifferent towards author B (i.e., intensity value is close to 0) then the intensity of the preference should be a small negative value (close to zero).

[1]Of course, there is also the explanation that the venue where A never published is extremely selective and author A's paper were just never accepted there. Both explanations are plausible but, since we do not have any information regarding the rejected submissions, we chose the one that can create negative preferences, in order to get a richer test workload.

6.2.2 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. For each user, we first select the list of authors and then we create, for each consecutive two preferences, one qualitative preference with intensity equal to the difference between the two quantitative preferences intensities. This mechanism will create qualitative preferences with negative, zero or positive intensity values. A zero intensity value for the qualitative preference means that the tuples resulted by applying any of the two preferences, that produce the qualitative preference, are equally preferred. The cases where the resulted intensity is a negative value are taken into account when we insert the preference in the preference graph. To avoid any negative intensity value in our preference graph, we reverse the order of the preferences and use the positive value instead. This is a perfectly correct mechanism since the strictly negative and strictly positive values are symmetric.

Proposition 7. *Let $\alpha > 0$ be the intensity value for the following qualitative preference: "A is preferred over B". Then the intensity value for the qualitative preference: "B is preferred over A" is $-\alpha$.*

Proof. Preference "A is preferred over B" with intensity $> 0 \Rightarrow$

$$intensity(A) > intensity(B) \tag{6.1}$$

and the strength of the preference is equal to α.

Let us assume that preference "B is preferred over A" has an intensity value, $\beta > 0 \Rightarrow$

$$intensity(B) > intensity(A), \forall \beta > 0 \tag{6.2}$$

and the strength of the preference is β.

However, Equation (6.2) contradicts the results given in Equation (6.1) which is the hypothesis. In this case, the assumption made is incorrect, therefore $\nexists\ \beta > 0$ such that "B is preferred over A" holds $\Rightarrow \beta < 0$ and, since the negative values and the positive values are symmetric, we can conclude that $\beta = -\alpha$ and preference "B is preferred over A" holds with intensity= $-\alpha$. □

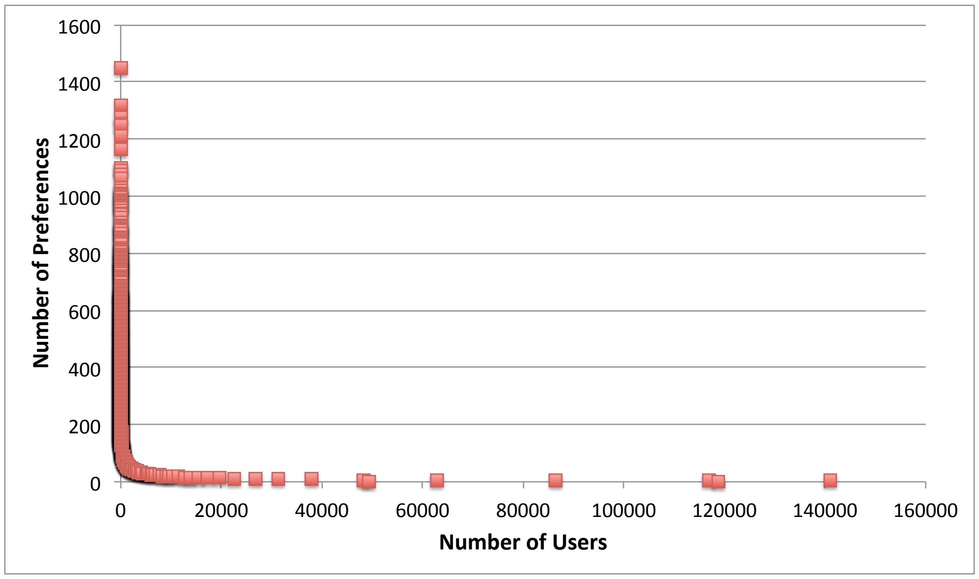


Figure 17: Distribution of Number of Preferences

To create these qualitative preferences we used the larger dataset of quantitative preferences (i.e., before removing the author preferences with intensity lower than 0.1). We made this decision because, in the case of qualitative preference, a zero intensity value means the tuples are equally preferred which is a valuable information.

The distribution of number of preferences extracted is presented in Figure 17. For all the preferences created we display on the Ox axis the number of users with the same number of preferences. The graph shows that there are only few users with a very high number of preferences, between 200 and 1500, and also with a very low number of preferences (i.e., 1, 2 preferences).

6.2.3 Conflict Resolution

When the preference graph is created, a new edge is introduced only if this does not create a conflict.

In our preference graph model there are two possible conflicts:

1. Conflicting behavior - Refers to the conflict that appears in the cases of the non-contextual preference representation, when the system inserts, in the graph, a new edge that creates a

cycle. In the base case, a cycle can be created between two nodes A, B if there is a directed edge from A to B and a new, directed edge from B to A is inserted. In order to see why this is considered a conflict, let us assume we have an edge, A $\rightarrow$ B, where A and B are two nodes in the graph. This edge means that the user prefers A more than B which implies that *intensity*(A) $>$ *intensity*(B). If there is another edge, B $\rightarrow$ A then, following the previous reasoning, *intensity*(B) $>$ *intensity*(A) in which case we reached a contradiction.

In a real system, this type of conflict can be solved in different ways.

- If the preference is provided online, the system can ask the user to specify which preference is more important.
- If user's feedback is not available, the edge can be marked as a conflict-edge until the conflict can be resolved (e.g., user provides a feedback or an intensity value is modified).
- The values provided by the user can be subtracted and the new value will be used to resolve the conflict.

In our model, we insert the conflicting edge but it labels it as *CYCLE*. When the graph is traversed, this edge is not taken into consideration.

2. Incompatible intensities – Refers to the conflict that appears in the case when an edge, A$\rightarrow$B, is introduced in the graph but nodes A and B already had assigned quantitative intensities and *intensity(A)* $<$ *intensity(B)*. This is a conflict because the directed edge implies that *intensity(A)* $>$ *intensity(B)*.

   In the case when both nodes, A and B, are connected to the graph we avoid this type of conflict by inserting the edge and labeling it *DROPPED*. In this case the edge will not be used when the graph is traversed, but it can be relabeled, and used later, if the preference intensities of the two involved nodes change.

   When one of the nodes has the in_degree (i.e., number of directed edges that comes into the node) or out_degree (i.e., number of directed edges that leave the node) equal to zero, we solve this conflict by re-computing the intensity value of this node and this way, we do not propagate the conflict.

Algorithm 7 Check conflict

Input: *Left node reference , Right node reference.*

Output: *FALSE, if no conflict is created by adding a new edge between leftNode and rightNode and both intensities are user provided; TRUE otherwise.*

```
1: BEGIN
2: if ( leftNode.intensity > rightNode.intensity AND (both intensities are user provided) ) then
3:     return FALSE
4: else
5:     return TRUE
6: end if
7: END
```

6.3 THE TIME COMPLEXITY OF CREATING THE HYPRE GRAPH

The Unified Preference Graph is created in two steps using the Algorithm 1. For each step, the system reads all preferences from the relational database and creates the necessary nodes and edges in the preference graph. When the list of quantitative preferences contains only unique preferences (i.e., unique SQL predicates), for each user we can insert all quantitative preferences in a batch, and there is no need to verify if there is a node, with that particular preference, inserted already in the graph. However, for the qualitative preferences, some of the nodes would have already been created (as quantitative preference) and therefore we need first to extract a reference to the node(s) in order to insert the qualitative preference. Because of that, we cannot take advantage of the batch insertion for the qualitative preference insertion step.

Table 11: Insertion Time

Insertion Type	Number of preference	Time (sec)
Quantitative Preferences	10,361,592	256.61
Qualitative Preferences	7,901,874	3680.26

Algorithm 8 Compute Intensity Value

Input:

LEFT/RIGHT : the position of the node for which the algorithm computes an intensity value, based on the direction of the edge

QT : intensity of a quantitative preference

QL : intensity of a qualitative preference

Output: System computed intensity value

```
1: BEGIN
2: if (LEFT) then
3:     return min(1, $QT * 2^{[sign(QT)*QL]}$) {/*compute intensity for the LEFT node*/ }
4: end if
5: if (RIGHT) then
6:     return max(-1, $QT * 2^{[-sign(QT)*QL]}$) {/*compute intensity for the RIGHT node*/ }
7: end if
8: END
```

Step1. The first step of the algorithm is to create all quantitative preferences. For this step, the algorithm reads preferences from the relational database and creates 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. We limited the batch size to 100,000 tuples because every batch insertion is considered one transaction and is kept in memory until the insertion is complete. Without this limit, the system runs out of memory because nothing is written on the disk until the transaction ends.

Step2. In the second step of the algorithm, the system inserts all qualitative preferences, for one user at a time, from the *qualitative_pref* table. Each row in this table contains a *leftPref* and a *rightPref* which are translated into two nodes in the preference graph, connected with a directed edge – from the left node to the right node. The left node receives the preference stored in *leftPref* attribute, and the right node acquires the preference stored in the *rightPref* attribute.

Each qualitative preference insertion is executed in one transaction. The "insertion" of one qualitative preference follows one of the three possible scenarios:

Scenario 1: Two nodes containing the *leftPref* and *rightPref* respectively are already in the user's subgraph. In this case, the algorithm only needs to insert an edge between the two nodes, and run a conflict check subroutine to ensure that no conflict in created by the new edge.

Scenario 2: Only one node containing one of the predicates is already in the user's subgraph. In this case, the algorithm creates a new node, adds an edge between the two nodes and computes an intensity value for the newly created node using the Algorithm 8 (based on Equation (4.1) and Equation (4.2)).

Scenario 3: The two new predicates are not already part of the user's subgraph. In this case, the algorithm creates the two nodes, adds the edge between them, assigns a DEFAULT_VALUE to one of the nodes and computes the value of the second one, using Algorithm 8. For this scenario we create two new nodes, that are not connected to any other nodes in the graph. Therefore, assigning a DEFAULT_VALUE to any of the two nodes will not create any conflict. However, if we assign the default value to the left node, the right node will receive a lower a value, computed using the Algorithm 8. In contrast, if we assign the default value to the right node, the left node will receive a value greater than the default value. The way a particular DEFAULT_VALUE is chosen is discussed later in Section 6.3.1.

Table 11 shows the time necessary to create the preference graph for all users. As expected, the insertion time for the quantitative preferences is much smaller than the insertion time for the qualitative preferences since the system can benefit from the batch insertion. However, for inserting almost 8 million qualitative preferences the system requires about an hour to finish.

6.3.1 Default Value Selection

The DEFAULT_VALUE in Algorithm 1 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 available. In the process of generating the preference graph we experimented with different values for this *seed*, as shown in Table 12. The first column con-

Table 12: Possible DEFAULT_VALUEs

Computing Algorithm	**Values Considered**	**Values Picked**
default	no condition	0.5
min	no condition	
min_pos	≥ 0	0
max	no condition	
max_pos	≥ 0 and < 1	0
avg	no condition	0.98
avg_pos	≥ 0	0

tains the name of the aggregate function we used to compute the DEFAULT_VALUE, the second column contains the values considered for one particular algorithm and, finally, the last column contains the value we assigned when none of the exiting intensity values matched the condition in the second column or if the returned value was equal to 1 (for avg case). Since the default value is a starting point, if this value is one, all values computed with this seed will be equal to one.

Except for the "default" algorithm that simply assigns a 0.5 value to the intensity, the DEFAULT_VALUE is computed for each user, individually, using the intensity values from his/her preference, 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 he/she already provided.

6.4 SUMMARY

In this chapter we described the preprocessing steps necessary to convert DBLP dataset into a list of user profiles in order to test our theoretical model. Our objective was to define preferences that are covering the qualitative and quantitative spectrum and are also meaningful.

We implemented our preference graph in an interactive 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 used Java for both creating and querying the graph database, and reading preferences and sending the enhanced query to the relational database. To communicate with RDBMS, we used a classical JDBC connection, whereas to query the graph database we used the embedded Java version of Neo4j and we sent Cypher queries to find the matching nodes and preferences respectively.

In the next chapter, we present the our system's behavior and we discuss the benefits and draw-backs of each algorithm designed in the previous chapter using the preferences extracted from DBLP-Citation-network dataset.

7.0 EXPERIMENTAL RESULTS

In this chapter we present the our system's behavior and we discuss the benefits and draw-backs of each algorithm, with emphasis on the variation of utility metric.

We implemented our preference graph in an interactive system, and we evaluate its practicality and usefulness under realistic conditions using the preferences generated in Chapter 6. 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.

The motivation of running each algorithm is to show the common difficulties that appear in a system that handles preferences. We show that intensity value plays a crucial role in determining the predicate combination order and we evaluate PEPS, our Top-*K* algorithm that selects the most preferred tuples based on the user profile.

7.1 EXPERIMENTAL RESULTS BASED ON UTILITY AND COVERAGE METRICS

In the next sections we apply the Utility and Coverage metrics defined in Section 5.1 to evaluate the performance of our proposed framework. Moreover, we show, by comparison, the behavior of different algorithms defined to combine preference predicates.

7.1.1 Experimental Results for Utility Metric

Equation (5.2) defines *Utility* metric as a product that depends on the number of tuples and the combined intensity value. Combinations that return significantly more tuples than any other combination but with a very small combined intensity value are considered outliers since the *Utility*

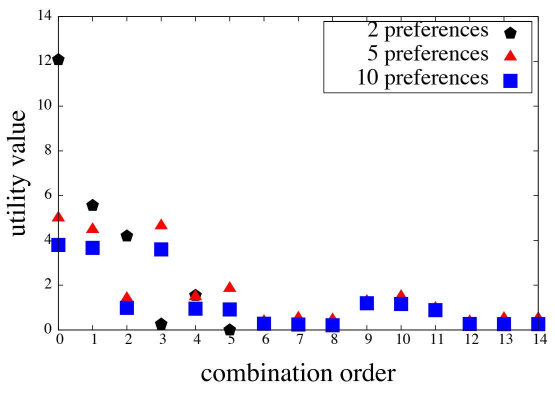


Figure 18: Utility Value (uid=2)

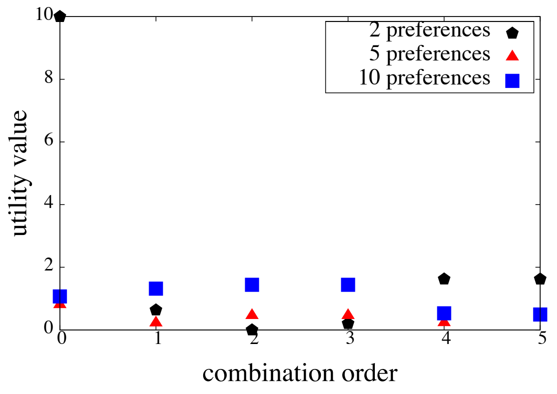


Figure 19: Utility Value (uid=38437)

metric is high for these tuples. To alleviate this problem we only took into account the first twenty-five tuples (i.e., the first page) from the result list.

Figure 18 and Figure 19 show the variation of the preference *Utility* for all combinations of two, five, and ten preferences for user with id=2 and id=38437, respectively. Because our algorithm reruns some of the preferences when a new preference is introduced with an AND operator, we have multiple times when we see a combination of 5 or 10 preferences. Moreover, more preferences we combine, more combinations we will have. Because of that, the X axis labeled "combination order" represents the order in which a combination of 2, 5 or 10 preferences was seen.

We can see that, as expected, there is an overall descending trend in utility value. However, although combining two preferences gives us the highest overall combined intensity value, in terms of utility, combination of two preferences is quickly topped by the combination of five preferences. In this case there is a tradeoff between the number of tuples returned and the intensity value attached to each tuple.

To better understand how the utility is changing we present next the variation of intensity and the number of tuples returned by each preference enhanced query for user with id=2.

In Figure 20, Figure 22, and Figure 24 we show the number of tuples returned for each combi-

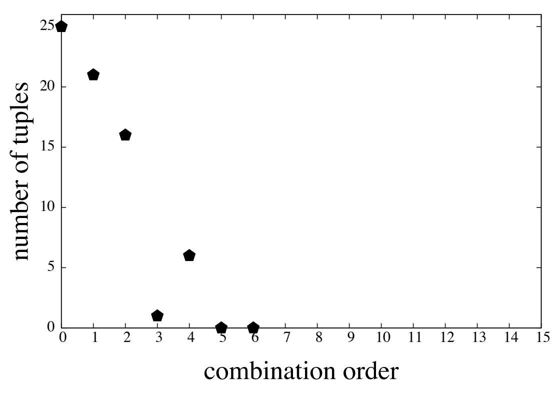


Figure 20: Number of Tuples for All Combinations of 2 Preferences

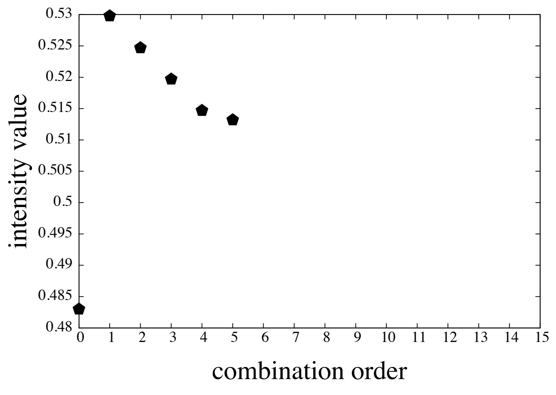


Figure 21: Intensity Value for All Combinations of 2 Preferences

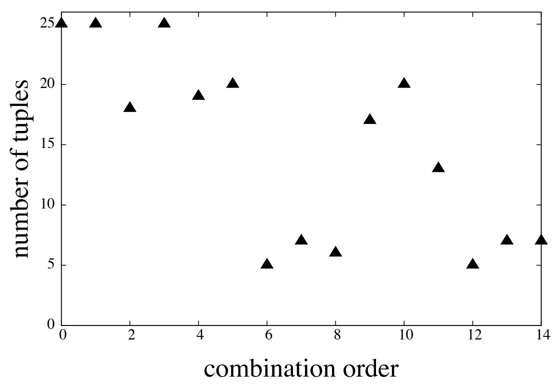


Figure 22: Number of Tuples for All Combinations of 5 Preferences

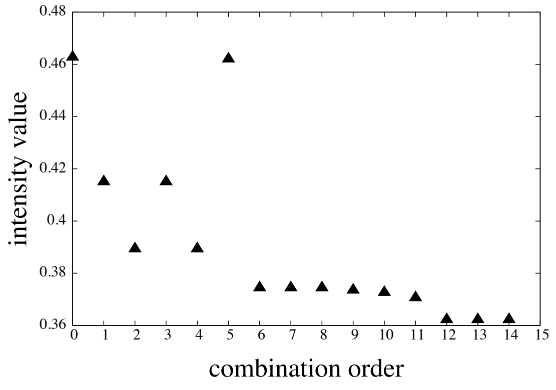
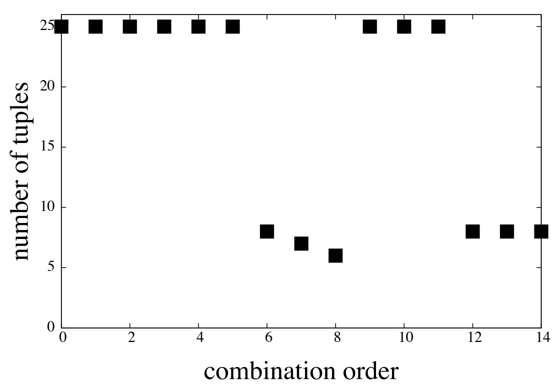


Figure 23: Intensity Value for All Combinations of 5 Preferences

nation of 2, 5 and 10 preferences respectively. Then, we show variation of the combined intensity value for the same combinations in Figure 21, Figure 23, and Figure 25, respectively.



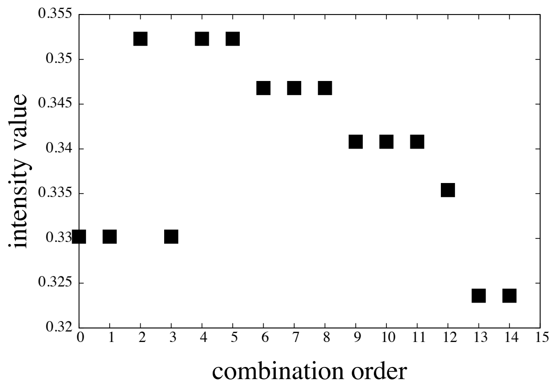


Figure 24: Number of Tuples for All Combinations of 10 Preferences

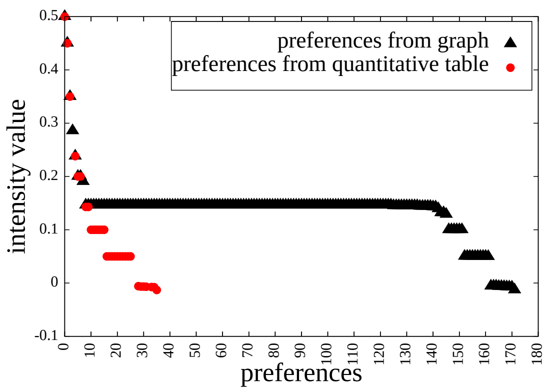
Figure 25: Intensity Value for All Combinations of 10 Preferences

7.1.2 Experimental Results for Coverage Metric

Our model is using intensity values to combine the two preference types which, in the end, generates significantly more quantitative preferences, as presented in Figure 26 and 27. For user with id=2, chosen at random, the graph shows that initially there are 36 quantitative preferences, but after inserting all qualitative preferences, the preference graph will contain 172 nodes. Similarly, for another randomly selected user, with id=38437, Figure 27 shows that the number of quantitative preferences, for user with uid=38437, increases from 24 to 50.

By using the formulas presented in Section 4.4, we are able to assign an intensity value to all nodes that are part of the qualitative preference by either generating a carefully selected default value or 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 into quantitative preferences, without losing the underlining information provided by a qualitative preference, that will still be stored in the graph.

With more quantitative preferences we can cover overall more tuples in the database. Figure 28



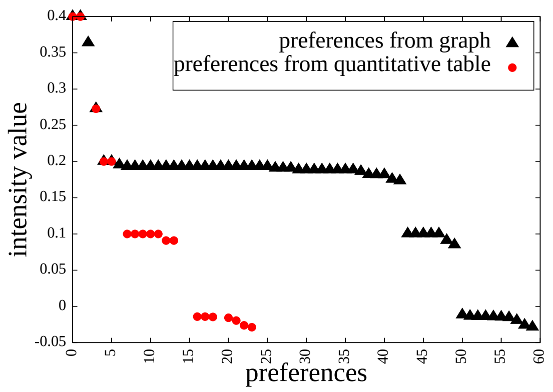


Figure 26: Variation of Number of Quantitative Preferences for uid=2

Figure 27: Variation of Number of Quantitative Preferences for uid=38437

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 UPG model

For the first case, qualitative preferences in the original form have only information about the intensity of the preference (i.e., 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. Figure 28 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 preferences. This improvement is from 120% compared to both quantitative and qualitative (uid=388437) up to 336% compared to just quanti-

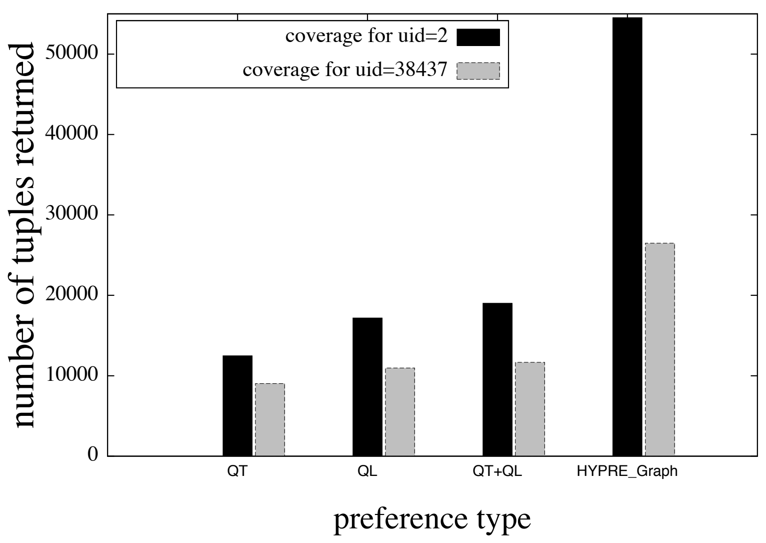


Figure 28: The Coverage Over the Dataset for uid=2 and uid=38437

tative 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.

7.2 DETERMINING THE BEST COMBINATION OF PREFERENCES

Quantitative preferences are very important in any system that tracks user's preferences because they facilitate ranking of tuples, from the most preferred to the least preferred, by assigning a score to each tuple that matches the user's preference. However, quantitative preferences are not as general as possible and, as we highlighted in Chapter 2, some cases cannot be supported by quantitative preferences only, and instead require qualitative preferences to express them. Our system uses intensity values to combine qualitative and quantitative preferences which not only creates a unified platform to store preferences, but also assigns intensity values for nodes that do not have any value (as in the qualitative case). In this way, all nodes in our graph can now be seen as quantitative preferences that, in turn, can be used to return tuples that match one or more preferences.

Ideally, we want to return tuples in descending intensity order. To do that, we can order the preferences based on their intensity and pick one or more preferences from this list. However, strictly ordering the preferences by their intensity value is not enough to return the set of tuples with the highest combined intensity as we illustrate in the following sections.

Before we provide an efficient solution to combine preferences such that the tuples with the highest intensity value will be returned first, we want to demonstrate the difficulty of determining such an order of combination. For this reason we designed three different types of algorithms.

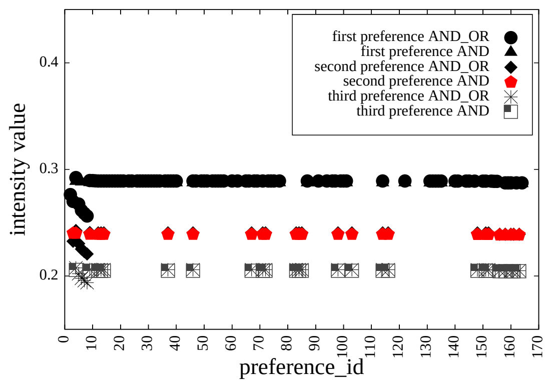
Clearly, the preference with the highest intensity will return the most preferred tuples. Moreover, when one tuple matches two different preferences, the final intensity value is a combination of the two individual intensity values. In Section 4.6 we discussed about the combination function used to compute the final intensity value and we mentioned that in the disjunctive combination case we use a *reserved* approach whereas, in the conjunctive combination case we consider the *inflationary* approach.

7.3 EXPERIMENTAL RESULTS FOR COMBINE-TWO ALGORITHM

We created the *Combine-Two* algorithm, described in Section 5.3.1, to demonstrate the complexity of preference combination along with the influence of intensity values on each combination. There are two flavors for this algorithm, however in both versions the algorithm combines only two preferences at a time.

- Version 1: It combines two preferences with an AND operator
- Version 2: It combines two preferences using AND operator for predicates with different attributes (e.g., *venue* and *author*) and OR operator for predicates referring to the same attributes

The results of running this algorithm are displayed in Figure 29 and Figure 31, for user with id=2 and Figure 30 for the user with id=38437, after all the combinations that return no tuples have been removed. In Figure 29 and Figure 30 we choose to display only the first three sets of



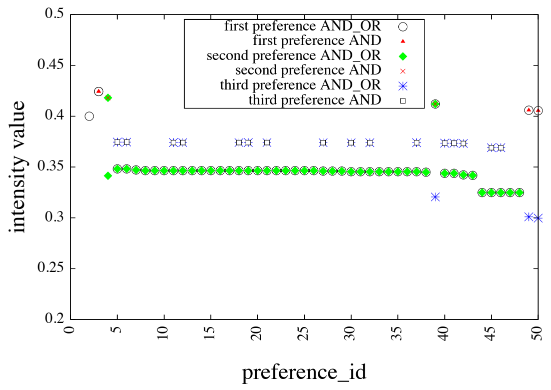


Figure 29: Variation of Intensity Value (uid=2) – *Combine-Two* Algorithm

Figure 30: Variation of Intensity Value (uid=38437) – *Combine-Two* Algorithm

combinations, first, second and third preference, respectively, combined with all preferences that follow. That is, the *first preference AND_OR* and *first preference AND* line in the graph represents the combination of the first preference with all the remaining preferences. The *second preference AND_OR* and *second preference AND* line represents the variation of intensity when combining the second preferences with the remaining preferences. Since preferences are presented in descending order by their intensity values, the overall intensity value decreases for every new step. The intensity variation is clear in the first 20 combinations and a closer look of this behavior is shown in Figure 31 for user with id=2.

There are two important things to see in this graph:

1. Combining the first preference with the third one returns tuples with a better intensity value than combining the first preference with the second one from the list, although the intensity value of the second preference is higher than the intensity value of the third preference (since preferences are sorted by intensity value). In the same way, combining the first preference with the eighth one returns a better intensity value than combining the first preference with all other preferences in between.

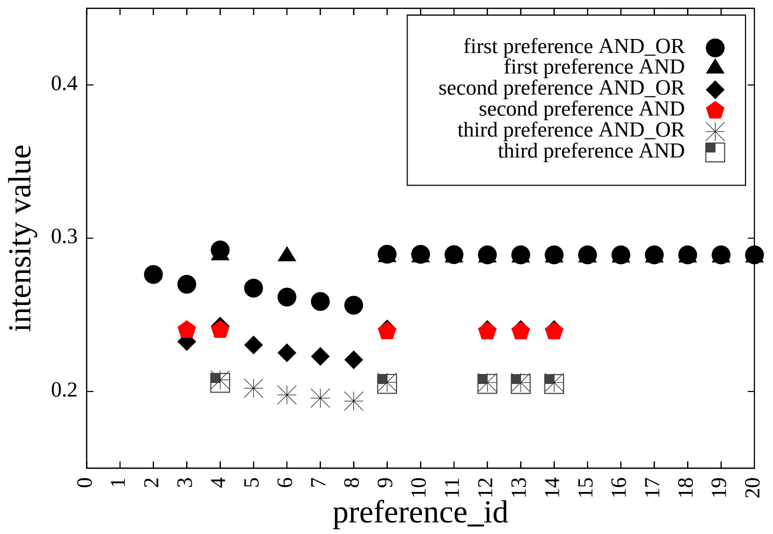


Figure 31: Intensity Value Variation for First 20 Combinations (uid=2) – *Combine-Two* Algorithm

2. Although AND semantics result in better intensity values when used, there are cases when the combined preference will not return anything (e.g., combining the first preference with the forth, sixth or seventh preference or combining the second and the third preference with al the preferences from 15 to 40). This happens either because the combinations of two given preferences do not return anything (e.g. the user has two author preferences, but this two authors never published together) or because of the information starvation problem discussed in Section 4.6.

This experiment shows that strictly ordering the preferences, in terms of their intensity values, is not enough to decide the order in which they should be combined. When we combine preferences using an AND operator, the final intensity value is larger than the two intensity values of the preferences that are combined. However, the combination might not return any tuple because the preference predicates are not compatible (e.g., two preferences on different venues) or because there is not enough data to satisfy the given preferences together (e.g., two preferences on different authors that have not published together, yet). On the other hand, when we combine preferences with an OR operator we are guaranteed to have tuples in the result (assuming the preferences do match at least one tuple in the database). However, the final intensity value lies between the two given intensity values. Because of this issues, the system cannot only look at the first few preferences, or all preferences with intensity value higher than a threshold, because an AND combination between the first preference and one of the last preference is still better than using the first preference alone.

7.4 EXPERIMENTAL RESULTS FOR PARTIALLY-COMBINE-ALL ALGORITHM

A preference-aware system needs to be able to combine all available preferences in order to decide which set of tuples are the most preferred, and not only two at a time. *Partially-Combine-All* algorithm, described in Section 5.3.2, is combining all preferences starting with the first one, using an AND operator between preferences for different attributes and an OR operator for preferences on the same attribute. Another important characteristic of this algorithm is that it reruns some old preferences with an addition of a new preference, as long as the new preference is introduced using an AND operator. We want to do this step because a combination with an AND operator always returns a combined intensity value higher than the two intensity values involved in the combination.

Figure 32 and Figure 34 show the results of this experiment for the user with uid=2, and Figure 33 shows the results for user with id=38437. Figure 32 shows the variation of intensity for all the times when the algorithm combines 2, 5 and 10 preferences whereas Figure 34 shows the variation of intensity for all combinations of 10 or more preferences. It is interesting to see that combining the first two preferences, with the highest intensity score does not return the highest combined intensity value, therefore is not the best option that one should pick for ranking the tuples. The same behavior can be seen for the 5 and 10 preferences combination which strengthens our hypothesis that intensity value and order of preference combination play a crucial role in determining which tuples are the most preferred. However, choosing to combine the preferences with the highest intensity values first does not necessarily return the most preferred tuples since combining the second best and the third best preference, with an AND operator, might return a

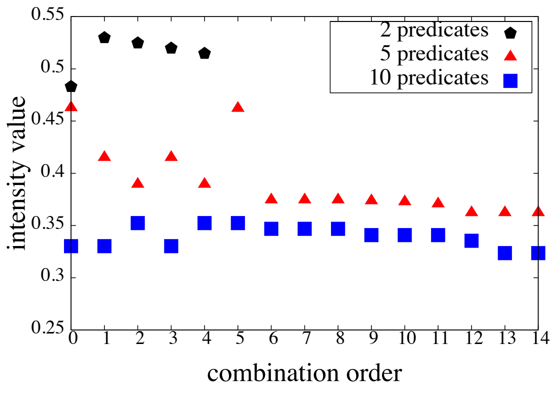


Figure 32: Intensity Value Variation (uid=2) - *Partially-Combine-All* Algorithm -All 2, 5 and 10 preferences

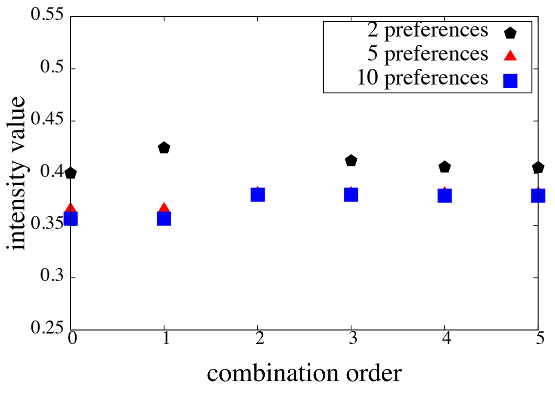


Figure 33: Intensity Value Variation (uid=38437) -*Partially-Combine-All* Algorithm -All 2, 5 and 10 preferences

higher intensity value then combining the first preference and the second, with an OR operator.

With this experiment, we showed that no matter how many preferences we combine – two at a time like in *Combine-Two* algorithm or more than two, like in *Partially-Combine-All* algorithm – there is still a problem of deciding which preferences to combine in order to maximize the combined intensity value and create applicable predicate combination.

7.5 EXPERIMENTAL RESULTS FOR BIAS-RANDOM ALGORITHM

The utility metric defined in Section 7.1.1 shows us that even though some combinations have high intensity value, because they do not return anything, their utility is equal to zero. Our system needs to be able to create combinations that will return tuples, without the need to explicitly run each combination, in order to optimize the running time of the algorithm. But without knowing which combinations return data, there is no efficient option to determine which partial combination is

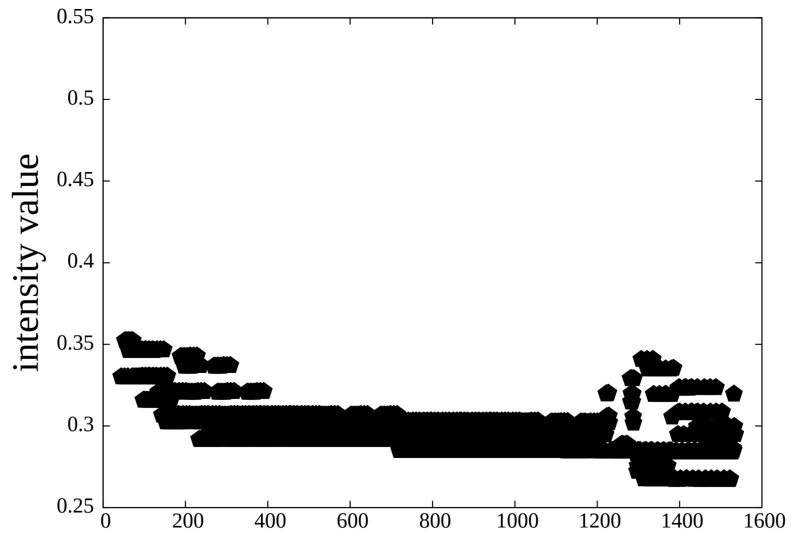


Figure 34: Intensity Value Variation (uid=2) -*Partially-Combine-All* Algorithm – 10 pref. or more

a solution that should be further pursued and which one should be dropped. We will show the difficulty of this selection later in this section.

To eliminate the run of each partial combination over the tuples in the dataset, the only solution is to create a list of all possible combinations, order them descending on their combined intensity value and use this list in order to filter the data. However, Proposition 4 from Section 5.2 proves that this solution is not feasible since it creates an exponential number of combinations.

We created the *Bias-Random* algorithm in order to show the complexity of deciding what combinations should be picked, and we described it in Section 5.4. The algorithm picks randomly a preference to be used in the preference combination. The selection is biased towards preferences with higher intensity values since these preferences are more important for a user. If the partial combination is a valid one (i.e., the cardinality of the result set is greater than zero) then the algorithm attempts to add a new randomly selected preference from the remaining set until nothing is returned anymore.

Since the algorithm has a random selection step, we ran the algorithm ten times, and we dis-

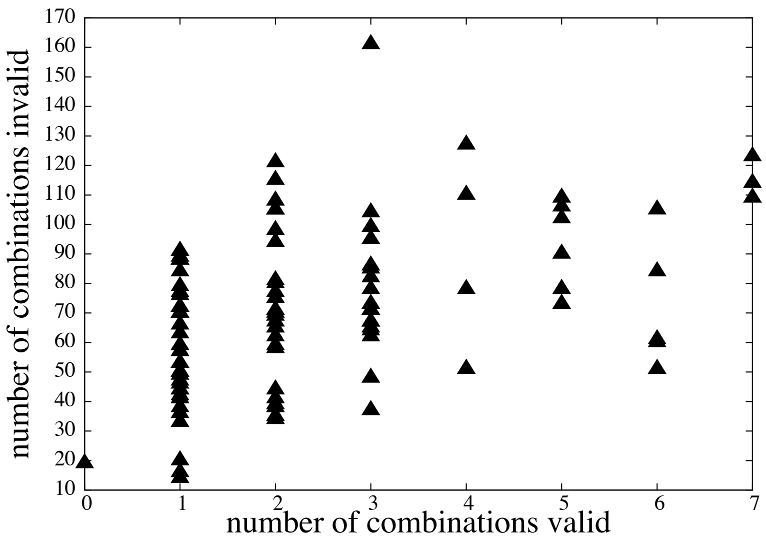


Figure 35: Number of Solutions vs. the Number of Different Combinations Checked (uid=2)

played for each run the number of non-valid combinations (i.e., those that return no results) along with the number of valid combinations. In Figure 35 and Figure 36 we present these results ordered by the number of valid solutions found for user with id=2 and id=38437, respectivelly.

In Figure 35, out of all 100 experiments ran, the best case happened when the algorithm tried around 30 non-valid combinations and 2 valid ones. However, in the worst case the algorithm tried 160 non-valid combinations and only 3 valid. Because a lot of combinations do not return anything, we need to efficiently pick only the valid ones, and we implemented a solution for this problem in our *Practical and Efficient Preference Selection* (PEPS) algorithm, presented in Section 5.5.

7.6 EXPERIMENTAL RESULTS FOR PEPS ALGORITHM

The *Practical and Efficient Preference Selection* (PEPS) algorithm, described in Section 5.5, is created, as the name states, to identify the preferences that return the best combined intensity value,

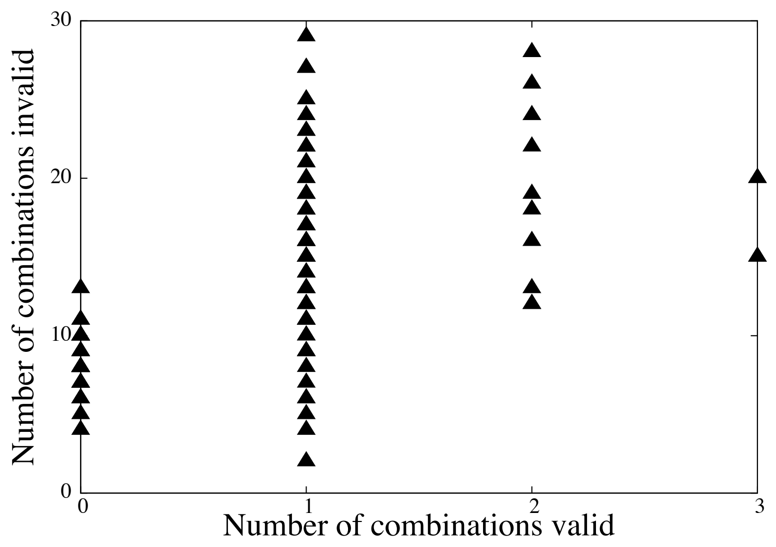


Figure 36: Number of Solutions vs. the Number of Different Combinations Checked (uid=38437)

in an efficient way, since this will also return the most preferred tuples when the query is enhanced with that particular preference. In order to evaluate our Top-K (PEPS) algorithm's correctness, we implemented the well-known Fagin's TA algorithm ([14]) by generating the combined intensity value for each paper, per user.

7.6.1 Top-K Baseline Algorithm

Fagin's TA algorithm assumes there are different scores (between 0 and 1) given for each tuple in the database, one score for each attribute. For *m* different attributes, the algorithm stores the tuples in *m* different list, ordered by their score on that particular attribute.

Definition 19. *TA intensity value – Let R be an object (i.e., tuple) in the database. If $x_1, \ldots, x_m$ (each in the interval [0,1]) are the grades of object R under the m attributes, then $t(x_1, \ldots, x_m)$ is the (overall) grade of object R.*

Definition 20. *Fagin's TA Algorithm – To retrieve the Top-K tuples based on their combined intensity value the algorithm executes the following steps:*

1. *Do sorted access in parallel to each of the m sorted lists Li. As an object R is seen under sorted access in some list, do random access to the other lists to find the grade x_i of object R in every list Li. Then compute the grade t(R)=$t(x_1, , x_m)$ of object R. If this grade is one of the k highest we have seen, then remember object R and its grade t(R) (ties are broken arbitrarily, so that only k objects and their grades need to be remembered at any time).*
2. *For each list Li, let x^i be the grade of the last object seen under sorted access. Define the threshold value τ to be $t(x^1, \ldots, x^m)$. As soon as at least k objects have been seen whose grade is at least equal to τ, then halt.*
3. *Let Y be a set containing the k objects that have been seen with the highest grades. The output is then the graded set $\{(R, t(R)), where R \in Y\}$.*

In our test dataset, we have "grades" on the *venue* and *author* attributes. 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 Equation (4.3).

For the intensity venue, we selected all quantitative preferences that refer to the venue attribute, and we extracted all tuples that match any of these preferences.

For the author attribute, because one paper usually has multiple authors, we computed a composite grade, using the $f_\wedge(p1, p2)$ formula for combining preferences with an AND operator, defined in Equation (4.3). In this way, we created an aggregate score for the *author* attribute.

The final step is to combine the two lists, and the final score for each tuple that is in both lists is again computed using the $f_\wedge(p1, p2)$ formula. Moreover, we also added all the tuples that are in only one list.

Before we present our comparison results, is it interesting to note that this algorithm is not scalable, since it necessitates a list of tuples for each attribute that has a preference defined on.

A system that is using this type of algorithm can be easily overwhelmed by the number of tables created (or by the number of times a pair <paper_id, intensity> is stored) since this process should be done for each user.

7.6.2 Similarity and Coverage Metrics

In order to better understand the similarities and differences between the results given by PEPS algorithms and the Fagin's TA algorithm we made use of two metrics, *Similarity* and *Overlap* defined below.

Definition 21. *Similarity – Given two lists of tuples, the similarity metric returns the percentage of tuples that are common in the two lists.*

The *Similarity* is the metric used to compare how close two lists are to each other. When the two lists contain all different tuples, the similarity is 0%, whereas when they contain exactly the same tuples, abstraction of the order, the similarity is 100%.

Definition 22. *Overlap – Given two lists with the same tuples, L1 and L2, the overlap metric returns the percentage of tuples that are in the same order in both lists.*

The *Overlap* metric is used to check the relative order of the tuples, when only the tuples that match in the two lists are compared. When the tuples appear in the same order in both lists, the overlap is 100%

7.6.3 Top-K Comparison Evaluation

The previous sections demonstrate the obstacles a system faces when it needs to decide which preferences, and in which order should be combined. Our *Practical and Efficient Preference Selection* Algorithms eliminate or alleviate some of these difficulties by using a pre-computed table of all combinations of two preferences.

Quantitative-only preferences: Because Top-K algorithms work only for the quantitative preferences, we first create a HYPRE graph that incorporates only the quantitative preferences. We ran our PEPS algorithm over this graph and we compared our results against those of the TA algorithm. The results show 100% similarity (i.e., the paper ids in the two lists with their combined

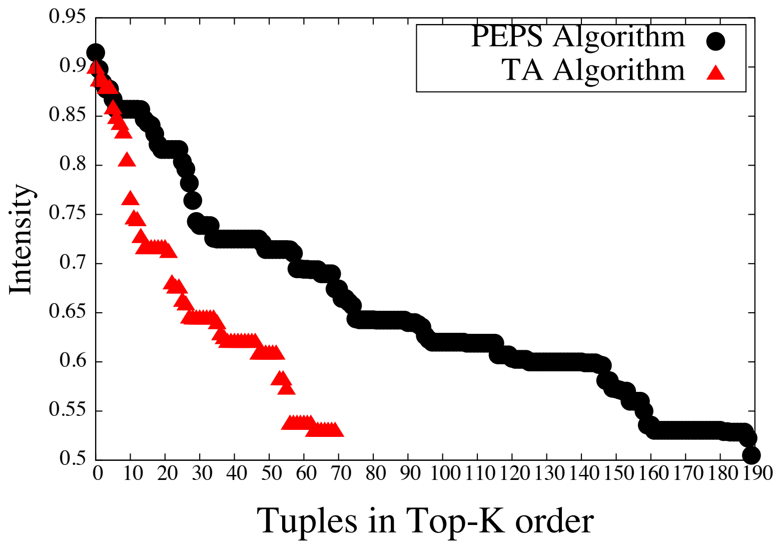


Figure 37: Variation of Intensity Value (uid=2) -*PEPS vs. TopK FA* Algorithm

intensity value match completely) and 100% overlap (i.e., the order of the paper ids in the final ranking match completely).

Both quantitative and qualitative preferences: In order to assess the advantages of PEPS when qualitative preferences are considered, we ran PEPS over the large HYPER 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) and user with uid=38437 (i.e., 0.4). The results depicted in Figure 37 and Figure 38 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 pref-

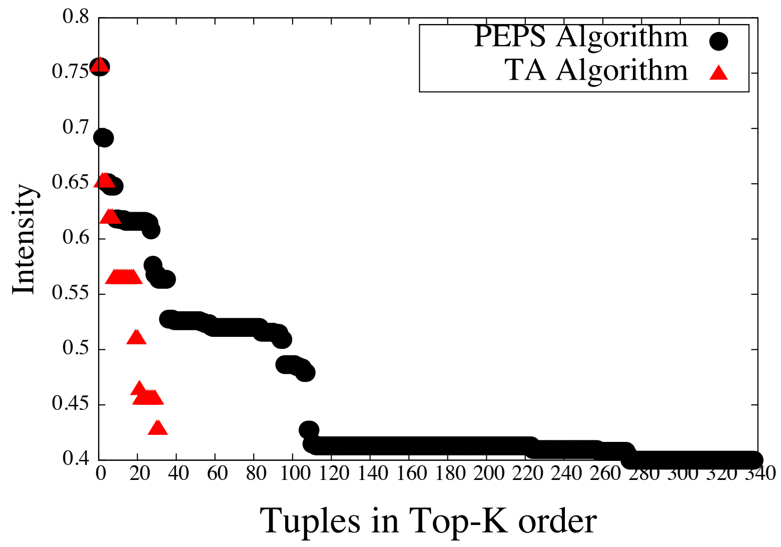


Figure 38: Variation of Intensity Value (uid=38437) -*PEPS vs. TopK FA* Algorithm

erence our system is assigning higher intensity values to the retrieved tuples than the quantitative preferences used by the TA algorithm.

When looking at the similarity between the two Top-K lists returned, we can see that there are only 37% matching tuples in the two lists, mostly because our system has access to more preferences (both qualitative and quantitative preferences) therefore can rank tuples that TA algorithm cannot (since the TA algorithm only works with quantitative preferences).

To measure the overlap between the two lists, we first extracted the 37% of matching tuples from the two lists and then we count the number of tuples for which the order is preserved across the two lists. We found that there is a 100% match between the two lists which allows us to say that, for the tuples that are common in the two list, both algorithms (PEPS and TA) are ordering them in the same way.

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,

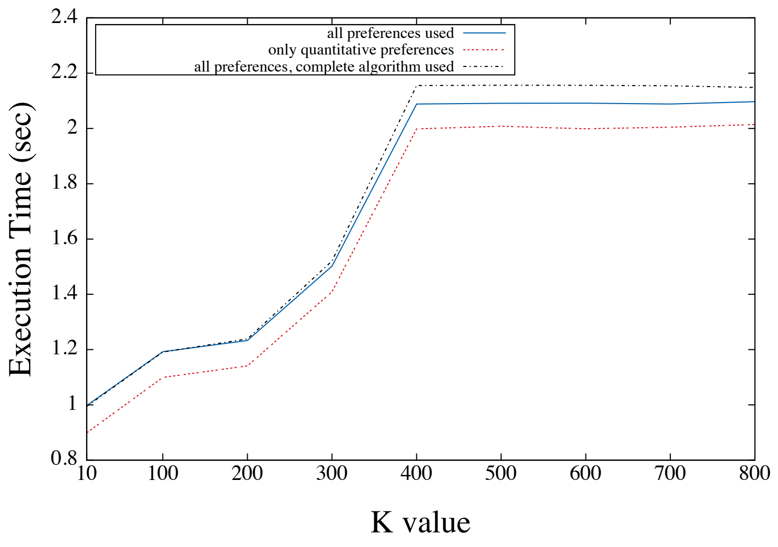


Figure 39: Time variation when size of K changes (uid=2) -*TopK PEPS* Algorithm

it does not incur any performance penalty. To measure the time complexity of our algorithm, we vary the size of K, from 10 to 800, in 100 increments, 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. We ran both the Approximate and Complete PEPS algorithm and even though the *complete* version is keeping all combinations that might create a valid combination, the execution time does not increase considerably. The averaged results are presented in Figure 39, for user with id=2 and in Figure 40 for user with id=38437, and we can see that for 800 tuples the Approximate PEPS algorithm only needs 2 seconds to run and the Complete PEPS algorithm only needs 2.2 seconds to run, for user with id=2 and 170 preferences and less than a second for the user with id=38437 and 50 preferences.

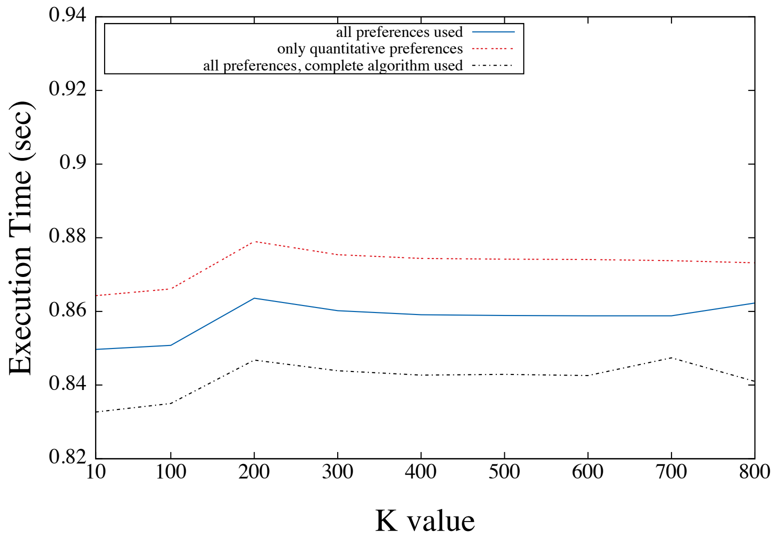


Figure 40: Time variation when size of K changes (uid=38437) -*TopK PEPS* Algorithm

8.0 CONCLUSIONS AND FUTURE WORK

8.1 SUMMARY OF CONTRIBUTIONS

In this dissertation we presented a new framework that incorporates qualitative and quantitative preferences in a hybrid, unified model using the notion of intensity. The notion of intensity captures the strength of a preference. Our model, HYPRE (*Hy*brid *Pre*ference) Graph, not only supports both types of preferences at the same time, but also associates intensity with both quantitative, as well as, qualitative preferences. The intensity value is used in two ways: on the one hand, it determines the order over the database tuples that matches different preferences, and, on the other hand, it is used to convert qualitative preferences into quantitative preferences without loosing the qualitative information. In this dissertation, we describe, in details, the theoretical model providing examples on how different types of preferences are supported. We also provide a special set of functions used to compute missing intensities values, which generate new intensity values based on the qualitative preference intensity value.

List of contributions:

1. In Chapter 3 we present our new model that incorporates qualitative and quantitative preferences into a unified hybrid preference framework which is based on a preference graph. A HYPRE Graph is a collection of user profiles that hold information about each user's preference. One preference can be one single node, in the case of quantitative preferences or a pair of nodes and a directed edge that connects them, for a qualitative preference. Moreover, any node in the graph holds information about the user id, the preference predicate and the intensity value associated with this predicate.
2. In Chapter 4 we describe the design and implementation of a real system prototype for our

hybrid model with the emphasis on the necessary functions designed to either assign intensity value to the nodes that do not have any value provided or (re-)compute intensity values to avoid any representational conflicts. Moreover, we describe the algorithms used to create and update the unified preference graph, while also detect and mark the conflicts.

3. In Chapter 5 we first introduce the *Coverage* and *Utility* metrics that are used to characterize the system's overall coverage over the tuples in the database and the influence of the individual intensity value and predicate selectivity over the final ranking of tuples based on the computed combined intensity value when predicates are combined either with OR operator or AND operator. Second, we discuss about different algorithms to combine preferences' predicates in terms of efficiency and time complexity. Our optimized Practical and Efficient Preference Selection algorithm overcomes the drawbacks introduced by randomly selecting preferences, or combining only two preferences at a time, or combining all available preferences and returns applicable predicate combinations in a relatively small amount of time.

4. In Chapter 6 we explain in-depth the necessary steps taken to convert a DBLP dataset with citation information into meaningful user profiles that cover both qualitative and quantitative preferences in different aspects.

5. Chapter 7 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.
   - 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.
   - We demonstrate that intensity plays a key role in determining the final ranking of the tuples and simple ordering the preferences by their intensity value is not enough to return a list of tuples ordered from the most to the least preferred.

8.2 FUTURE WORK

The work presented in this dissertation can be extended on different levels.

First, we can combine a predicate-based preference graph with an attribute-based preference graph. This way, using appropriate algorithms that converts an attribute-based preference into an SQL query, we will be able to run Skyline queries over the database that will return a better approximation of "the best" tuples when an exact result does not exist.

Second, each preference is stronger if the context for which it was define is collected and used. A context based preference graph is a natural extension to HYPRE graph introduced in this dissertation. Moreover, having context information, some of the conflicts will be resolved since a conflicting situation in HYPRE graph can be break into multiple non-conflicting preferences depending on the context when they are applicable.

Third, our system can be enhanced with different algorithms that extract and collect preferences to alleviate the burden on the user side. Combining multiple profiles into a group (e.g., all users working in the database group at University of Pittsburgh) a system can have access to more preferences and recommend items using the collective list of preferences. This is especially important in the case when a user does not have too many preferences. Moreover, a system could possibly watch user's behavior and use the feedback received to update some of the intensity values already defined.

8.3 IMPACT OF THIS DISSERTATION

The impact of this dissertation can be characterized based on two different areas:

1. **Impact in the science and technology**. The solution provided in this dissertation creates the base of a new model that combines and extends different previously studied areas like database preference, graph preferences and ranking. From this point of view, this dissertation advances in three different aspects:
   - New paradigm: This dissertation introduces a new, hybrid preference graph model that combines two different type of preferences – *qualitative* and *quantitative* preferences by

converting qualitative preferences into quantitative ones without losing the qualitative information. The new model and records, for each preference, its intensity value and uses these values to assign intensity values to preferences that do not have a predefined value. Any preference system works better when the strength of the preference is provided, that is why the quantitative model is used more in real life applications.

- Information overloading: With our model we attempt to give a solution for the personalization problem that allows a system to rank the tuples based on a predefined user profile. In an era of Big Data, being able to extract fast, and filter out tuples that important for one particular user are mandatory characteristics for any system that deals with continuously increasing amount of data.
- Preference representation: Using the basic definition of a preference graph previously introduced in the literature, we were able to expend it into a new model that stores together all user profiles with a fast insertion and retrieval of all preferences for one particular user.

2. **Impact to society**. Our society is strongly connected to data and retrieval of meaningful information from a sea of available options is a requirement in every aspect of our life. Many times users find themselves in an uncomfortable position. They want or need to find a piece of information but they end up "asking" many questions in order to find what they are looking for or, in the worst case, they end up with an empty list because their questions were too strict. Being able to store a user profile in a HYPRE Graph, a system can have a fast access to the most preferred tuples and less effort is requested from the user, if any. This will impact the user-friendliness of any application that deals with big amount of data.

BIBLIOGRAPHY

[1] R. Agrawal, R. Rantzau, and E. Terzi. Context-sensitive ranking. In *Proceedings of the 2006 ACM SIGMOD International Conference on Management of data*, SIGMOD'06, pages 383–394, 2006.

[2] R. Agrawal and E. L. Wimmers. A framework for expressing and combining preferences. In *SIGMOD 2000*, pages 297–306, 2000.

[3] J. Akbarnejad, G. Chatzopoulou, M. Eirinaki, S. Koshy, S. Mittal, D. On, N. Polyzotis, and J. S. V. Varman. Sql querie recommendations. *Proc. VLDB Endow.*, 3:1597–1600, September 2010.

[4] L. Bellatreche, A. Giacometti, P. Marcel, H. Mouloudi, and D. Laurent. A personalization framework for olap queries. In *Proceedings of the 8th ACM international workshop on Data warehousing and OLAP*, DOLAP '05, pages 9–18. ACM, 2005.

[5] J. Bentham. *An Introduction to the Principles of Morals and Legislation (Collected Works of Jeremy Bentham).* Clarendon Press, 1996.

[6] C. Boutilier, R. I. Brafman, C. Domshlak, H. H. Hoos, and D. Poole. Cp-nets: a tool for representing and reasoning with conditional ceteris paribus preference statements. *Journal of Artificial Intelligence Research*, 21(1):135–191, feb 2004.

[7] J. Cho and S. Roy. Impact of search engines on page popularity. In *WWW'04, Proceedings of the 13th international conference on World Wide Web*, pages 20–29. ACM, 2004.

[8] J. Chomicki. Querying with intrinsic preferences. In *Proceedings of the 8th International Conference on Extending Database Technology: Advances in Database Technology*, EDBT '02, pages 34–51, 2002.

[9] J. Chomicki. Preference formulas in relational queries. *ACM Trans. Database Syst.*, 28(4):427–466, Dec. 2003.

[10] P. Ciaccia. Querying databases with incomplete cp-nets. In *Multidisciplinary Workshop on Advances in Preference Handling*, M-PREF'07, 2007.

[11] S. Cohen and M. Shiloach. Flexible xml querying using skyline semantics. *ICDE 2009*, 0:553–564, 2009.

[12] C. Domshlak, E. Hüllermeier, S. Kaci, and H. Prade. Preferences in ai: An overview. *Artificial Intelligence*, 175(7-8):1037–1052, 2011.

[13] M. Endres and W. Kiessling. Transformation of tcp-net queries into preference database queries. In *Proceedings of the ECAI 2006 Multidisciplinary Workshop on Advances in Preference Handling*, pages 23–30, 2006.

[14] R. Fagin, A. Lotem, and M. Naor. Optimal aggregation algorithms for middleware. *Journal of Computer and System Sciences*, 66(4):614 – 656, 2003. Special Issue on {PODS} 2001.

[15] P. W. Foltz and S. T. Dumais. Personalized information delivery: An analysis of information filtering methods. *Commun. ACM*, 35(12):51–60, Dec. 1992.

[16] P. Georgiadis, I. Kapantaidakis, V. Christophides, E. M. Nguer, and N. Spyratos. Efficient rewriting algorithms for preference queries. In *Proceedings of the 24th International Conference on Data Engineering*, ICDE'08, pages 1101 –1110, 2008.

[17] R. Gheorghiu, A. Labrinidis, and P. K. Chrysanthis. Database preferences -a unified model. *PersDB*, 2012.

[18] A. Hey, S. Tansley, and K. Tolle. *The fourth paradigm: data-intensive scientific discovery*. Microsoft Research, 2009.

[19] S. Holland and W. Kiessling. Situated preferences and preference repositories for personalized database applications. In *Proceedings of the 23rd International Conference on Conceptual Modeling*, volume 3288 of *Lecture Notes in Computer Science*, pages 511–523, 2004.

[20] R. Kambalakatta, M. Kumar, and S. K. Das. Profile based caching to enhance data availability in push/pull mobile environments. *Mobile and Ubiquitous Systems, Annual International Conference on*, 0:74–83, 2004.

[21] W. Kiessling. Foundations of preferences in database systems. In *Proceedings of the 28th international conference on Very Large Data Bases*, VLDB'02, pages 311–322, 2002.

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

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

[24] G. Koutrika and Y. Ioannidis. Personalization of queries in database systems. In *Data Engineering, 2004. Proceedings. 20th International Conference on*, pages 597 – 608, 2004.

[25] G. Koutrika and Y. Ioannidis. Personalized queries under a generalized preference model. In *ICDE 2005*, pages 841–852, 2005.

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

[27] M. Lacroix and P. Lavency. Preferences; putting more knowledge into queries. In *Proceedings of the 13th International Conference on Very Large Data Bases*, VLDB'87, pages 217–225, 1987.

[28] J. J. Levandoski, M. F. Mokbel, and M. Khalefa. Flexpref: A framework for extensible preference evaluation in database systems. In *ICDE 2010*, pages 828 –839, march 2010.

[29] F. Liu, C. Yu, and W. Meng. Personalized web search for improving retrieval effectiveness. *IEEE Trans. on Knowl. and Data Eng.*, 16(1):28–40, Jan. 2004.

[30] J. Manyika, M. Chui, B. Brown, J. Bughin, R. Dobbs, C. Roxburgh, and A. H. Byers. Big data: The next frontier for innovation, competition, and productivity. *McKinsey Global Institute*, May 2011.

[31] A. Miele, E. Quintarelli, and L. Tanca. A methodology for preference-based personalization of contextual data. In *Proceedings of the 12th International Conference on Extending Database Technology: Advances in Database Technology*, EDBT'09, pages 287–298, 2009.

[32] D. Mindolin and J. Chomicki. Hierarchical cp-networks. In *Proceedings of the Third Multidisciplinary Workshop on Advances in Preference Handling*, M-PREF'07, 2007.

[33] S. Mobasher and B. Lytinen. Concept Based Query Enhancement in the ARCH Search Agent. In *Proceedings of the 4th International Conference on Internet Computing (IC'03)*, June 2003.

[34] P. Neophytou, R. Gheorghiu, R. Hachey, T. Luciani, D. Bao, A. Labrinidis, E. G. Marai, and P. K. Chrysanthis. Astroshelf: Understanding the universe through scalable navigation of a galaxy of annotations. *Proceedings of Special Interest Group of Management of Data*, pages pp. 1–4, 2012.

[35] E. Pitoura, K. Stefanidis, and P. Vassiliadis. Contextual database preferences. *IEEE Data Eng. Bull.*, 34(2):19–26, 2011.

[36] P. Roocks, M. Endres, S. Mandl, and W. Kiessling. Composition and efficient evaluation of context-aware preference queries. In *DASFAA (2)*, pages 81–95, 2012.

[37] D. Skoutas, M. Alrifai, and W. Nejdl. Re-ranking web service search results under diverse user preferences. In *PersDB 2010, Personalized Access, Profile Management and Context Awareness in Databases*, 2010.

[38] D. Souravlias, M. Drosou, K. Stefanidis, and E. Pitoura. On novelty in publish/subscribe delivery. In *ICDEW 2010*, pages 20–22, 2010.

[39] K. Stefanidis, M. Drosou, and E. Pitoura. Perk: personalized keyword search in relational databases through preferences. In *EDBT 2010, 13th International Conference on Extending Database Technology, Lausanne, Switzerland, March 22-26,*, pages 585 –596. ACM, 2010.

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

[41] K. Stefanidis, E. Pitoura, and P. Vassiliadis. Modeling and storing context-aware preferences. In *Advances in Databases and Information Systems*, pages 124–140, 2006.

[42] K. Stefanidis, E. Pitoura, and P. Vassiliadis. Adding context to preferences. In *ICDE*, pages 846–855. IEEE, 2007.

[43] J. Stoyanovich, W. Mee, and K. A. Ross. Semantic ranking and result visualization for life sciences publications. In *ICDE 2010*, pages 860–871, 2010.

[44] 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*, 2011.

[45] A. Tversky and D. Kahneman. The framing of decisions and the psychology of choice. *Science*, 211(4481):453–458, 1981.

[46] A. H. van Bunningen, L. Feng, and P. M. Apers. A context-aware preference model for database querying in an ambient intelligent environment. In *Database and Expert Systems Applications*, pages 33–43, 2006.

[47] A. Ventresque, S. Cazalens, T. Cerqueus, P. Lamarre, and G. Pasi. Personalization through query explanation and document adaptation. In *PersDB 2010, Personalized Access, Profile Management and Context Awareness in Databases*, 2010.

[48] M. Yakout, A. K. Elmagarmid, and J. Neville. Ranking for data repairs. In *ICDEW 2010*, pages 23–28, 2010.