Bulletin of the Technical Committee on

Data Engineering

June 2011 Vol. 34 No. 2 **IEEE Computer Society**

Letters

Special Issue on Personalized Data Management

Conference and Journal Notices

Letter from the Editor-in-Chief

David Lomet
Microsoft Corporation

Letter from the Special Issue Editor

Personalization takes data management to a new frontier where each user is able to get a tailored service according to the personal preferences, behavior, and surrounding context. This special issue includes ten articles that address various aspects of personalization, namely, preference queries, context-aware data management, and recommender systems.

The issue starts by two articles that lay out the foundation of preference queries in relational database systems. Preference queries present a major component in personalized data management where two different users querying a database using the same query may receive different results based on their personal preferences. The first article by Jan Chomicki gives a formal framework in which user preferences are formulated using first-order logic. The article also discusses the use of that logic in preference query evaluation and optimization in relational database systems. The second article by Werner Kießling *et al.* gives an overview of the Preference SQL system; a declarative extension of standard SQL by strict partial order preferences. Preference SQL enables a seamless preference application integration with SQL back-end systems.

The following article by Evaggelia Pitoura *et al.* calls for enhancing preference queries by considering the contextual information. Context may express conditions on situations external to the database or related to the data stored in the database. The article goes on outlining a model for expressing preferences and context to provide a more personalized query answer.

The fourth article in this issue by Georgia Koutrika identifies the differences between plug-in and native realization of preference queries in database management systems. This article aims at showing how tightly preferences are currently coupled with database queries and sharing a vision regarding opportunities and challenges in fully implementing preferences as first-class citizens inside the database engine by changing both the database query model and internal code.

The issue then includes three articles presenting system prototypes for preference and context-aware systems. Yannis Ioannidis *et al.* presents the PAROS system that offers personalized services to its users through a user model, profiling strategies which create instances of personalized user models, and adaptation strategies that adapt the system behavior based on the user profiles. Justin Levandoski *et al.* presents the CareDB system; a context and preference-aware database system that includes a generic and extensible query processing engine, a framework for handling expensive attributes, and a framework for supporting uncertain data. Cristiana Bolchini *et al.* gives an overview of various research related to context modeling and awareness within the Context-ADDICT project.

The last part of this issue includes three articles about recommender systems, which mainly aim to provide content that is likely to interest users, based on current and/or past user behavior. Gloria Chatzopoulou *et al.* presents the QueRIE system that recommends to its users a set of queries that can be posed to the underlying database system. Such system is mostly useful to those users who lack SQL expertise or familiarity with the database schema, e.g., users from the scientific community. Mohammad Khabbaz *et al.* presents the TopRecs^{+} system that extends current recommender systems from only recommending a single item to recommend package of items and to consider user-specified constraints. The issue is then concluded by an article from Sihem Amer-Yahia giving an overview of various projects at Yahoo! Labs in the context of recommendations within web search.

I sincerely hope that you will enjoy reading this issue, and find it interesting and thought-provoking.

Mohamed F. Mokbel
Department of Computer Science and Engineering,
University of Minnesota
Minneapolis, MN, USA

Logical Foundations of Preference Queries

Jan Chomicki
University at Buffalo
chomicki@buffalo.edu

Abstract

The notion of preference plays an increasing role in today's information systems. In particular, preferences are used to specify which query answers are the best from the user's point of view. In this article, we discuss the work done over the last 10 years in the context of preference queries to relational database systems. We focus on preferences that are specified logically. *We show how such preferences can be embedded into relational query languages. By separating the preferences from the query, preference-specific query evaluation and optimization techniques can be formulated and preference modification studied. We conclude with an outline of future prospects for preference research.*

1 Introduction

Preference is one of the dimensions of personalization. Two different users querying a database using the same query may expect different answers because they may use different, implicit criteria of which answers are the *best* and the *most preferred*. For example, when purchasing a book online one user may be primarily interested in obtaining the lowest price available, while another may be concerned with the reliability of the vendor.

We discuss here a formal framework in which user preferences are formulated explicitly using first-order logic. Relying on that representation, many issues germane to preferences like composition, elicitation, or revision have been studied independently of queries. Moreover, it has been shown that preferences and queries can be combined in a clean fashion, yielding *preference queries*. In the context of such queries, classical database issues like query evaluation and optimization have been revisited, yielding new, preference-specific techniques.

Preferences have been studied for a long time in decision theory and philosophy [Fis70, Han01]. The interest in preferences in artificial intelligence [BD09] and databases [SKP11] is more recent.

The following mock car-shopping dialogue illustrates some aspects of preferences and preference queries addressed in this paper:

Maggie (salesperson): What kind of car do you prefer?

Fred (customer): The newer the better, if it is the same make. And cheap, too.

Maggie: Which is more important for you: the age or the price?

Fred: The age, definitely.

Maggie: Those are the best cars, according to your preferences, that we have in stock.

Fred: Wait...it better be a BMW.

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

First, Maggie elicits Fred's preferences involving age and price. Second, she gets Fred to commit to a *prioritized composition* of those preferences (age more important than price). Third, Maggie runs a *preference query* returning the best cars, according to Fred's preferences. Fourth, Fred changes his mind and *revises* his preferences.

2 Preference relations

We view preferences as *binary relations* between objects of the same kind. *Attribute* preference relations relate attribute values (typically, constants), *tuple* preference relations relate tuples in the same relation. To refer to the former we use the symbol $''>''$, to the latter, the symbol $''\succ''$, both with subscripts if necessary. If $x \succ y$, we say that x is *is preferred to* y (x *is better than* y, x *dominates* y).

Finite preference relations are defined by enumerating their elements. To define infinite preference relations – common in the presence of infinite domains – we use *first-order logic formulas.*

Example 1: Throughout this paper, we will repeatedly use the database relation $Car(Make, Year, Price)$ and the following preference relations:

$$p >_{price} p' \equiv p < p'$$
$$y >_{age} y' \equiv y > y'$$
$$(m,y,p) \succ_{C_1} (m',y',p') \equiv m = m' \wedge (y > y' \wedge p \leq p' \vee y \geq y' \wedge p < p').$$

The first two are attribute preference relations, the third, a tuple preference relation.

Typically, logic formulas defining preferences (*preference formulas*) contain constants, variables, comparison operators (like $''>''$) and Boolean connectives. Thus, it is possible to check whether one tuple is preferred to another by substituting tuple attribute values into the preference formula and computing the truth value of the resulting formula. Such preferences – based only on the contents of the tuples being compared – are called *intrinsic* [Cho03], in contrast to *extrinsic* preferences which may also refer to the contents of database relations. Also, intrinsic preference formulas usually admit quantifier elimination, and thus quantifiers are not needed in preference specification.

We also make use of several *derived* binary relations:

- non-strict attribute preference: $x \geq_A x' \equiv x >_A x' \vee x = x'$;
- non-strict tuple preference: $t \succeq_C t' \equiv t \succ_C t' \vee t = t'$;
- tuple indifference: $t \sim_C t' \equiv t \not\succ_C t' \wedge t' \not\succ_C t$.

2.1 Strict partial orders

Here we list some typical properties of binary relations. A binary relation R is

- *irreflexive* if $\forall x\ (\neg R(x,x))$,
- *transitive* if $\forall x,y,z\ (R(x,y) \wedge R(y,z) \rightarrow R(x,z))$,
- *connected* if $\forall x,y\ (R(x,y) \vee R(y,x) \vee x = y)$,
- a *strict partial order* (SPO) if it is irreflexive and transitive,

- a *weak order* (WO) if it is an SPO such that $\forall x,y,z\ (R(x,y) \rightarrow R(x,z) \vee R(z,y))$,
- *a total order* if it is a connected SPO.

Commonly, preference relations are required to be SPOs. It is obvious that irreflexivity should hold: preferring an object over itself seems to violate the basic intuitions behind preference. But transitivity is debatable. On one hand, it captures the *rationality* of preferences [Fis70, Fis99]. On the other, transitivity is sometimes violated by preference aggregation in voting scenarios [SP69]. We note that quantifier elimination provides procedures for checking whether a binary relation satisfies the desirable order properties.

Preferences are often captured using numeric-valued *scoring functions*. Such functions constitute special cases of preference relations. Indeed, a scoring function f represents a preference relation $\succ_f$ such that

$$x \succ_f y \equiv f(x) > f(y).$$

It is easy to see that preference relations represented by scoring functions are *weak orders*. Suppose $x \succ_f y$. Then $f(x) > f(y)$. So for every z, $f(x) > f(z)$ or $f(z) > f(y)$, and thus $x \succ_f z$ or $z \succ_f y$. The WO property has an important consequence: preference relations that are SPOs but not weak orders *cannot be represented using scoring functions*. Such preferences are common: the preference relation $\succ_{C_1}$ in Example 1, which is a *skyline* preference relation, falls into that category.

2.2 Combining preferences

There are many different ways in which preferences can be combined. We discuss *preference composition* and *preference accumulation*. Both of them are defined *logically*: if the preference relations being combined are defined by logic formulas, so is the resulting preference relation.

Preference composition combines preference relations about objects of the *same kind*: constants or tuples from the same database relation. The result is a preference relation of that kind. Therefore, the "dimensionality" of preference is not increased. The most common composition operators [1] are:

- union: $x \succ y \equiv x \succ_1 y\ \vee\ x \succ_2 y$, similarly intersection;
- prioritized composition: $x \succ y \equiv x \succ_1 y \vee (y \not\succ_1 x \wedge x \succ_2 y)$;
- Pareto composition: $x \succ y \equiv (x \succ_1 y \wedge y \not\succ_2 x) \vee (x \succ_2 y \wedge y \not\succ_1 x)$.

One of the potential applications of preference composition is *preference revision* [Cho07a]. This is further described in Section 6.

Example 2: The notation $X \succ Y$ means that $\forall x \in X, y \in Y\ (x \succ y)$. Consider two preference relations $\succ_1$ and $\succ_2$: $wine \succ_1 \{tea, coffee\} \succ_1 juice$ and $\{tea, juice\} \succ_2 coffee \succ_2 wine$. Then the prioritized composition $\succ_3$ of $\succ_1$ and $\succ_2$ is $wine \succ_3 tea \succ_3 coffee \succ_3 juice$ and the Pareto composition $\succ_4$ of $\succ_1$ and $\succ_2$ is $tea \succ_4 \{coffee, juice\}$. According to the indifference relation corresponding to $\succ_4$, wine and all the other drinks are mutually indifferent.

Preference accumulation combines preferences over *objects* to yield preferences over Cartesian products of objects. In this way, the "dimensionality" of preference is increased. The most common accumulation operators are:

- prioritized accumulation $\succ = \succ_1 \,\&\, \succ_2$: $(x_1,x_2) \succ (y_1,y_2) \equiv x_1 \succ_1 y_1 \vee (x_1 = y_1 \wedge x_2 \succ_2 y_2)$, and
- Pareto accumulation $\succ = \succ_1 \otimes \succ_2$: $(x_1,x_2) \succ (y_1,y_2) \equiv (x_1 \succ_1 y_1 \wedge x_2 \succeq_2 y_2) \vee (x_1 \succeq_1 y_1 \wedge x_2 \succ_2 y_2)$.

We note that both prioritized and Pareto accumulation are *associative* (Pareto is also commutative).

[1] The preference relation which is the result of the composition will be denoted by $\succ$.

3 Skylines

Among all preference relations, *skyline* preference relations, defined using Pareto accumulation, have been the most extensively studied [BKS01]. Given attribute preference relations $>_{A_1}, \ldots, >_{A_n}$, the *skyline* preference relation $\succ$ is defined as:

$$\succ = >_{A_1} \otimes >_{A_2} \otimes \cdots \otimes >_{A_n} .$$

Unfolding the definition of Pareto accumulation:

$$(x_1, \ldots, x_n) \succ (y_1, \ldots, y_n) \equiv \bigwedge_i x_i \geq_{A_i} y_i \wedge \bigvee_i x_i >_{A_i} y_i. \tag{1}$$

If we fix the attribute preferences to be the standard orderings of the reals, then we get a *Euclidean* skyline preference relation.

Example 3: Given a two-dimensional Euclidean skyline preference relation $\succ$:

$$(x_1, x_2) \succ (y_1, y_2) \equiv x_1 \geq y_1 \wedge x_2 > y_2 \vee x_1 > y_1 \wedge x_2 \geq y_2$$

and a finite set of points S in the 2-dimensional space, the skyline consists of $\succ$-maximal elements of S. Figure 1 shows an example skyline (the skyline points are solid black).

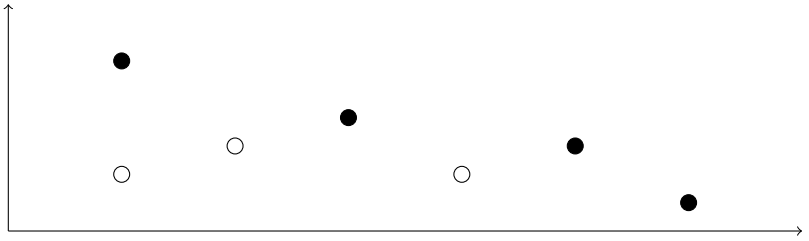

Figure 1: Two-dimensional skyline

Skyline preference relations and skylines enjoy several properties that make them attractive in the context of preference queries:

- *invariance*: a skyline preference relation is unaffacted by *scaling* or *shifting* in any dimension;
- *universality*: a skyline consists of maxima of *monotonic* scoring functions.

We note that usually skyline preference relations are not weak orders.

Example 4: In two-dimensional Euclidean space:

$$(3,0) \succ (2,0), (3,0) \not\succ (1,1), (1,1) \not\succ (2,0).$$

Actually, the skyline preference relations in [BKS01] admit a form slightly more general than the formula in Equation 1. The preference relation is defined not only in terms of attribute preference relations but also attribute equality. This achieves the effect of GROUP-BY and can be conceptually viewed as defining multiple skylines.

Example 5: Considering Example 1, the preference relation $\succ_{C_1}$ is defined as

$$(m, y, p) \succ_{C_1} (m', y', p') \equiv m = m' \wedge (y > y' \wedge p \leq p' \vee y \geq y' \wedge p < p').$$

Only the cars of the same make can be compared (are in the same group). In the SQL extension proposed in [BKS01], the above preference relation is expressed as

```
SKYLINE Make DIFF, Year MAX, Price MIN
```

Skylines have also been generalized to *p-skylines*, defined using not only Pareto but also prioritized accumulation [MC11b]. In this way priorities between different attribute preferences are captured.

We conclude this section by noting that all the preferences that can be expressed in the framework of Kießling [8] can also be expressed using intrinsic first-order logic formulas.

4 Preference queries

The most common kind of preference query has been formalized as the *winnow* operator ω [Cho03], also called *BMO* [8] and *Best* [TC02]. The operator retrieves all the best (undominated) tuples from a given relation.

Formally, given a preference relation $\succ$ and a database relation r:

$$\omega_{\succ}(r) = \{t \in r \mid \neg\exists t' \in r.\, t' \succ t\}.$$

If a preference relation $\succ_C$ is defined using a formula C, then we write $\omega_C(r)$, instead of $\omega_{\succ_C}(r)$.

It is clear that a skyline is the result of computing winnow under the skyline preference relation. This simple observation has important consequences: the techniques for evaluating and optimizing winnow apply immediately to skylines.

Winnow seamlessly integrates with the operators of the relational algebra. In fact, winnow can be expressed in relational algebra and provides the nonmonotonic functionality equivalent to that of set difference [Cho03].

Example 6: Consider the preference relation $\succ_{C_1}$ from Example 1, and the following relation instance $\{t_1 : (mazda, 2009, 20K), t_2 : (ford, 2008, 15K), t_3 : (ford, 2007, 15K)\}$ Winnow returns the tuples t_1 and t_2. The tuple t_3 is dominated by t_2.

There are two algorithms that compute winnow for arbitrary SPO preference relations: BNL [BKS01] and SFS [CGGL03]. Those algorithms were first proposed in the context of skylines but they only require irreflexivity and transitivity of preferences [Cho03]. Many algorithms for computing skylines and their variants have been proposed in the literature: we mention three influential ones: BBS [PTFS05], LESS [GSG07], and Salsa [BCP08].

5 Query optimization

A major advantage of the logic-based approach to preference queries is that *rewrite-based query optimization* is done in a natural and clean way. For example, the algebraic laws involving winnow are formulated analogously to the well-known laws of relational algebra [8, Cho03, HK05]. However, often the laws do not hold unconditionally. Consider commuting winnow and selection. We have $\sigma_C(\omega_\gamma(r)) = \omega_\gamma(\sigma_C(r))$ for every r if the formula $\forall t_1, t_2.[C(t_2) \wedge \gamma(t_1, t_2) \Rightarrow C(t_1)$ is valid.

Example 7: Under the preference relation $\succ_{C_1}$ from Example 1, the selection $\sigma_{Price<20K}$ commutes with ω_{C_1} but $\sigma_{Price>20K}$ does not.

Other laws involving winnow were studied in [Cho03, HK05].

Semantic query optimization (query optimization using integrity constraints) also fits in very well here. As shown in [Cho07b], the information about integrity constraints can be used to eliminate redundant occurrences of winnow and make more efficient computation of winnow possible. We say that ω_C is *redundant w.r.t.* a set of integrity constraints F if $w_C(r) = r$ for all r satisfying F. Now ω_C is *redundant w.r.t. F* iff F implies the formula $\forall t_1, t_2.\ R(t_1) \wedge R(t_2) \Rightarrow t_1 \sim_C t_2$. The latter formula is a *constraint-generating dependency*. The properties of such dependencies were studied in [BCW99].

6 Prospects for preferences

Preference modification Preferences are rarely static. Desires and needs fluctuate, priorities change. Several different preference modifications operators have been considered in the literature. In *preference revision* [Cho07a], a *revising* preference relation $\succ_0$ is *composed* with the original preference relation $\succ$ using one of the composition operators: union, prioritized or Pareto composition. Under certain restrictions on preference relations, the composition eliminates preference conflicts. Moreover, to guarantee SPO properties the result of the composition is transitively closed. Preference revision is suitable in scenarios where new preference information augmenting or contradicting the existing one comes to light. In *preference contraction* [MC11a], the new information consists of a *contractor* relation *CON*. The revised preference relation is now a maximal SPO subset of $\succ$ disjoint with *CON*. Contraction is appropriate if preferences are being cancelled or withdrawn. In *substitution* [BGS06], the new information is a set of *indifference* pairs. The paired objects are supposed to become mutually substitutable. This is possible if additional preferences are added so that the paired objects have the same sets of dominating and dominated objects. Clearly, having a general framework encompassing the above approaches (and other that can be envisioned) would be useful.

Preference elicitation. While preference formulas concisely and precisely capture preferences, they are not easy to construct. Instead of requiring that the user build a preference formula from scratch, one could imagine a step-by-step preference specification process in which the user could provide additional feedback. Recent work suggests that the feedback in the form of good or bad objects may be used in a variety of ways. In [JPL$^+$08], the user analyzes the result of a skyline query and labels some of the skyline objects as *superior* (should be in the skyline) or *inferior* (should not be in the skyline). Then the attribute preference relations are revised in such a way that the revised skyline query is guaranteed to return the superior objects and not to return inferior objects. Under the same model, [MC11b] propose that it is the skyline query expression that needs to be revised by replacing some occurrences of Pareto accumulation operators by the occurrences of prioritized accumulation. In this way relative priorities of attribute preference relations are captured. It would be natural to consider other kinds of user feedback, for example answers to dominance queries: *"does a dominate b?"* In general, it is a considerable challenge to uncover the preferences underlying the kinds of preference-related information available, for example, in social networks.

Preference and uncertainty. Several papers have studied the evaluation of skyline queries over uncertain (probabilistic) data [PJLY07]. It would be interesting to consider the same problem for more general variants of skyline queries. [KML08] study the problem of skyline querying in the presence of nulls. There, a tuple t_1 dominates a tuple t_2 if t_1 is *known* to be better than t_2 in some dimension and *not known* to be worse in any other dimension. What is the right logic for defining such preference relations?

Preferences over sets. In some applications it is natural to consider preferences over *sets of objects*. Those sets may be *homogenous* (consisting of objects of the same type) or *heterogenous* (consisting of objects of different types). Preferences over homogenous sets arise, for example, in committee selection, or employee or student recruiting. To address this topic, [ZC11] (generalizing the approach of [BBDS09]) propose a two-layered approach. In the first layer, *set profiles*, which are tuples of *features*, are defined. An example feature is an aggregate value of some attribute of the set, e.g., `SUM`. In the second layer, tuple preferences among profiles are specified. Queries return the best profiles – such profiles correspond to the best subsets of a given set. The algorithmic challenge is to improve on the brute-force enumeration of all the subsets. Preferences over heterogenous sets arise in product configuration. An example product is a vacation package, consisting of hotel, plane, and rental reservations. The best products are computed using the skyline semantics [WWI$^+$09]. The algorithmic challenge is to avoid the materialization of all possible products.

Preference networks. We note that an influential approach to preference queries and personalization [KI10] is based on the notion of *preference network* in which numeric scores are associated with query conditions to capture their degree of satisfaction. It would be interesting to develop logical semantics for that approach.

References

[BBDS09] Maxim Binshtok, Ronen I. Brafman, Carmel Domshlak, and Solomon Eyal Shimony. Generic Preferences over Subsets of Structured Objects. *Journal of Artificial Intelligence Research*, 34:133–164, 2009.

[BCP08] Ilaria Bartolini, Paolo Ciaccia, and Marco Patella. Efficient Sort-Based Skyline Evaluation. *ACM Transactions on Database Systems*, 33(4), 2008.

[BCW99] M. Baudinet, J. Chomicki, and P. Wolper. Constraint-Generating Dependencies. *Journal of Computer and System Sciences*, 59:94–115, 1999. Preliminary version in ICDT'95.

[BD09] R. I. Brafman and C. Domshlak. Preference Handling – An Introductory Tutorial. *AI Magazine*, 30(1):58–86, 2009.

[BGS06] W-T. Balke, U. Güntzer, and W. Siberski. Exploiting Indifference for Customization of Partial Order Skylines. In *International Database Engineering and Applications Symposium (IDEAS)*, pages 80–88, 2006.

[BKS01] S. Börzsönyi, D. Kossmann, and K. Stocker. The Skyline Operator. In *IEEE International Conference on Data Engineering (ICDE)*, pages 421–430, 2001.

[CGGL03] J. Chomicki, P. Godfrey, J. Gryz, and D. Liang. Skyline with Presorting. In *IEEE International Conference on Data Engineering (ICDE)*, pages 717–719, 2003.

[Cho03] J. Chomicki. Preference Formulas in Relational Queries. *ACM Transactions on Database Systems*, 28(4):427–466, December 2003.

[Cho07a] J. Chomicki. Database Querying under Changing Preferences. *Annals of Mathematics and Artificial Intelligence*, 50(1-2):79–109, 2007.

[Cho07b] J. Chomicki. Semantic Optimization Techniques for Preference Queries. *Information Systems*, 32(5):660–674, 2007.

[Fis70] P. C. Fishburn. *Utility Theory for Decision Making*. Wiley & Sons, 1970.

[Fis99] P. C. Fishburn. Preference Structures and their Numerical Representations. *Theoretical Computer Science*, 217:359–383, 1999.

[GSG07] P. Godfrey, R. Shipley, and J. Gryz. Algorithms and Analyses for Maximal Vector Computation. *VLDB Journal*, 16:5–28, 2007.

[Han01] S. O. Hansson. Preference Logic. In D. Gabbay, editor, *Handbook of Philosophical Logic*, volume 4. Kluwer, 2001.

[HK05] Bernd Hafenrichter and Werner Kießling. Optimization of Relational Preference Queries. In *Australasian Database Conference (ADC)*, pages 175–184, 2005.

[JPL+08] Bin Jiang, Jian Pei, Xuemin Lin, David W. Cheung, and Jiawei Han. Mining Preferences from Superior and Inferior Examples. In *KDD*, pages 390–398, 2008.

[KI10] G. Koutrika and Y. E. Ioannidis. Personalizing Queries Based on Networks of Composite Preferences. *ACM Transactions on Database Systems*, 35(2), 2010.

[Kie02] W. Kießling. Foundations of Preferences in Database Systems. In *International Conference on Very Large Data Bases (VLDB)*, pages 311–322, 2002.

[KML08] Mohamed E. Khalefa, Mohamed F. Mokbel, and Justin J. Levandoski. Skyline Query Processing for Incomplete Data. In *IEEE International Conference on Data Engineering (ICDE)*, pages 556–565, 2008.

[MC11a] D. Mindolin and J. Chomicki. Contracting Preferences for Database Applications. *Artificial Intelligence*, 175(7-8):1092–1121, May 2011. Special issue on Representing, Processing, and Learning Preferences.

[MC11b] D. Mindolin and J. Chomicki. Preference Elicitation in Prioritized Skyline Queries. *VLDB Journal*, 20(2):157–182, 2011. Special issue: selected papers from VLDB 2009.

[PJLY07] Jian Pei, Bin Jiang, Xuemin Lin, and Yidong Yuan. Probabilistic Skylines on Uncertain Data. In *VLDB*, pages 15–26, 2007.

[PTFS05] D. Papadias, Y. Tao, G. Fu, and B. Seeger. Progressive Skyline Computation in Database Systems. *ACM Transactions on Database Systems*, 30(1):41–82, 2005.

[SKP11] K. Stefanidis, G. Koutrika, and E. Pitoura. A Survey on Representation, Composition and Application of Preferences in Database Systems. *ACM Transactions on Database Systems*, 36(4), 2011.

[SP69] A. K. Sen and P. K. Pattanaik. Necessary and Sufficient Conditions for Rational Choice under Majority Decision. *Journal of Economic Theory*, 1:178–202, 1969.

[TC02] R. Torlone and P. Ciaccia. Which Are My Preferred Items? In *Workshop on Recommendation and Personalization in E-Commerce*, May 2002.

[WWI+09] Qian Wan, Raymond Chi-Wing Wong, Ihab F. Ilyas, M. Tamer Özsu, and Yu Peng. Creating competitive products. *PVLDB*, 2(1):898–909, 2009.

[ZC11] X. Zhang and J. Chomicki. Preference Queries over Sets. In *IEEE International Conference on Data Engineering (ICDE)*, pages 1019–1030, April 2011.

The Preference SQL System – An Overview

Werner Kießling, Markus Endres, Florian Wenzel
University of Augsburg, Institute for Computer Science
86135 Augsburg, Germany
{*kiessling, endres, wenzel*}*@informatik.uni-augsburg.de*

Abstract

Preference SQL is a declarative extension of standard SQL by strict partial order preferences, behaving like soft constraints under the BMO query model. Preference queries can be formulated intuitively following an inductive constructor-based approach. Both qualitative methods like e.g. Pareto / skyline and quantative methods like numerical ranking, definable over categorical as well as numerical attribute domains can be used. The Preference SQL System is implemented as a middleware component, enabling a seamless application integration with standard SQL back-end systems. The preference query optimizer performs algebraic transformations of preference relational algebra as well as cost-based algorithm selection e.g. for efficient Pareto / skyline evaluation. Ongoing work extends Preference SQL towards efficient support for personalized location-based mobile geo-services and social networks.

1 Introduction

After years of intensive experiences in particular through the Internet, search engine technology has improved considerably. However, more often than desirable it still does not meet the user's expectations. Prominent deficiencies which probably all search engine users have suffered are the infamous "empty-result" effect (i.e., getting no query results) and its opposite companion the notorious "flooding effect" (i.e., getting far too many query results). Thus a typical user search session requires several trial-and-error style query attempts, until eventually something suitable has been found, or the user quits frustratedly. Ad hoc attempts to remedy this unpleasant situation such as parametric search can achieve some relief, but do not solve the intrinsic problems. Focusing on search engines based on SQL databases, the cause for these difficulties is easily identified: SQL only supports hard constraints with an exact-match semantics. In contrast, mostly people also include preferences in their decisions, both in their daily lives as well as in business. But the very nature of preferences is different from hard constraints. Preferences are like soft constraints, requiring a match-making process instead: If my favorite choice is available in the database, I will take it. Otherwise, instead of getting nothing, I am open to alternatives, but show me only the best ones available in the database. Therefore, improving SQL-based search capabilities asks for extending SQL query technology towards a preference model, which of course should be powerful, flexible and intuitive for the user, but simultaneously query performance must be fast enough. However, the question of "what is the right preference model" is far from being trivial. In fact, the complexity of preferences has challenged researchers from diverse disciplines. During the past fifty years preferences have traditionally

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

been studied in fields such as Economic Decision Making, Social Choice Theory, Operations Research and Artificial Intelligence (see [3] for an overview). In recent years preferences have found steadily increasing interest in databases as well. A survey on the state of the art in databases can be found in [15].

In this paper we adopt the preference model introduced by [8], suggesting preferences to be strict partial orders (SPO). A similar approach has been proposed by [4]. SPO preferences can be interpreted in a very intuitive manner as personalized wishes in the form of "I like A more than B". Formally, a *preference P* on a set of attributes A is defined as $P := (A, <_P)$, where $<_P \subseteq dom(A) \times dom(A)$ is a strict partial order. $x <_P y$ is interpreted as "I like y more than x". Note that the preference order $<_P$ is irreflexive and transitive. An important subclass of preferences are *weak order preferences* (WOP), i.e., SPOs for which negative transitivity holds: $\forall x,y,z \in dom(A)$: $\neg(x <_P y) \wedge \neg(y <_P z) \Rightarrow \neg(x <_P z)$. For a WOP $P = (A, <_P)$ the dominance test can be efficiently performed by a numerical *utility function* $f : dom(A) \rightarrow \mathbb{R}_0^+$ which depends on the type of preference. Dominated tuples have higher function values, i.e., $x <_P y \iff f(x) > f(y)$. Note that this preference definition does not explicitly take the user's situation or context into account (see [12] for a discussion on context-awareness). Instead, preference-driven applications are supposed to maintain a persistent preference repository in order to automatically retrieve proper preferences for context-dependent query composition (see [7] for details).

Based on this preference model a first version of Preference SQL, extending SQL queries by preferences, has been described in [12]. An early commercial implementation of Preference SQL was available already in 1999. For the sake of rapid prototyping and quick time to market this version implemented loose coupling to a standard SQL database together with a query re-writing approach. Since then state of the art for preference handling in database systems has advanced considerably (see [15]). Here we provide an overview on the recent version of Preference SQL developed at the University of Augsburg.

2 The Preference Query Model

2.1 Preference Constructors

For the design of the declarative preference query language in [8] an inductive, constructor-based approach was used to formally specify a preference. For this purpose, a choice of intuitive *base preference constructors* together with some *complex preference constructors* has been defined, all being SPOs.

Base Preferences Preferences on single attributes are called *Base preferences*. There are base preference constructors for *continuous* and for *discrete (categorical)* domains. Figure 1 shows the taxonomic "ISA"-hierarchy of several frequently occuring base preference constructors.

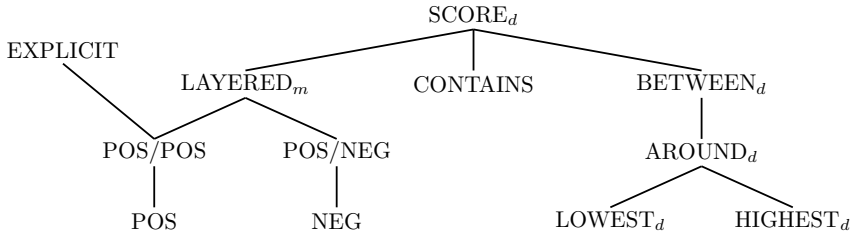

Figure 1: Taxonomy of base preference constructors

Definition 1 (SCORE$_d$ Preference): Given a scoring function $f_d : dom(A) \rightarrow \mathbb{R}$, and some $d > 0$. Then P is called a SCORE$_d$ preference, iff for $x,y \in dom(A)$: $x <_P y \iff f_d(x) > f_d(y)$ where $f_d(v), v \in dom(A)$, is defined as: $f_d(v) = \lceil f(v)/d \rceil$. Note that $f(v)$ is a numerical utility function.

When dealing with numerical values, it is a common practice to group ranges of scores together. Such a real-world behavior can be modeled using a d-parameter. It maps different function values to a single number which therefore maybe become substitutable with other values.

Since SCORE_d specifies a WOP, all sub-constructors of SCORE_d in Figure 1 are WOPs, too. However, EXPLICIT is not a WOP, hence not a sub-constructor of SCORE_d. EXPLICIT is used to create any preference that can be expressed by a finite set of "A is better than B" relationships. The continuous preference constructor BETWEEN_d expresses the wish for a value between a *lower* and an *upper* bound.

Definition 2 (BETWEEN_d Preference): The BETWEEN_d(A, [low, up]) preference is a SCORE_d preference with the following scoring function for $low, up \in dom(A)$:

$$f(v) := \begin{cases} low - v & ,\text{if } v < low \\ 0 & ,\text{if } low \leq v \leq up \\ v - up & ,\text{if } up < v \end{cases}$$

Specifying $low = up$ $(=: z)$ in BETWEEN_d we get the AROUND_d(A, z) preference constructor. In the AROUND_d(A, z) preference the desired value should be z. If this is infeasible, values within a distance of d are acceptable. Furthermore, specializing $z = \inf_A$ and $z = \sup_A$ yield the constructors LOWEST_d(A) and HIGHEST_d(A), resp., where $\inf_A$ / $\sup_A$ are the infimum / supremum of $dom(A)$. HIGHEST_d and LOWEST_d allow users to express easily their desire for values as high or as low as possible.

Definition 3 (LAYERED_m Preference): Let $L = (L_1, \cdots, L_{m+1})$ $(m \geq 0)$ be an ordered list of $m+1$ sets forming a partition of $dom(A)$ for an attribute A. The preference P is a LAYERED_m preference if it is a SCORE_0 preference with the following scoring function: $f(x) := i - 1 \iff x \in L_i$. For convenience, one of the L_i may be named "others", representing the set $dom(A)$ without the elements of the other subsets.

All categorical base preferences are sub-constructors of LAYERED_m, e.g. POS(A, POS-set) is equal to LAYERED_1(A, POS-set, others), and expresses that a user has a set of preferred values, the POS-set, in the domain of A. There is also a NEG-preference NEG(A, NEG-set). Moreover, it is possible to combine these preferences to POS/POS or POS/NEG. For the POS/POS(A, POS1-set, POS2-set) preference a desired value *should be* amongst a finite set POS1-set. Otherwise it *should be* from a disjoint finite set of *alternatives* POS2-set. If this is also not feasible, better than getting nothing any other value is acceptable. In addition to the discussed preferences we support the base preference CONTAINS which works on text attributes and currently supports simple full-text search. For details on all preferences we refer to [8, 10].

Complex Preferences If one wants to combine several preferences into an overall preference, one has to decide the relative importance of these given preferences. Intuitively, people speak of "this preference is more important to me than that one" or "these preferences are all equally important to me". We model equal importance of preferences by the so-called Pareto preference, whereas for ordered importance we introduce prioritization. A generalization to more than two preferences is straightforward, cp. [10].

Definition 4 (Pareto Preference): Assume preferences $P_1 = (A_1, <_{P_1})$, $P_2 = (A_2, <_{P_2})$, and $x = (x_1, x_2)$, $y = (y_1, y_2) \in dom(A)$, then the Pareto preference constructor denoted by $P := P_1 \otimes P_2$ is defined as:

$$(x_1, x_2) <_P (y_1, y_2) \text{ iff } (x_1 <_{P_1} y_1 \wedge (x_2 <_{P_2} y_2 \vee x_2 = y_2)) \vee (x_2 <_{P_2} y_2 \wedge (x_1 <_{P_1} y_1 \vee x_1 = y_1))$$

Definition 5 (Prioritization): Assume two preferences $P_1 = (A_1, <_{P_1})$ and $P_2 = (A_2, <_{P_2})$, then prioritization denoted by $P := P_1 \,\&\, P_2$ is defined as: $(x_1, x_2) <_P (y_1, y_2)$ iff $x_1 <_P y_1 \vee (x_1 = y_1 \wedge x_2 <_{P_2} y_2)$.

Generalizing above definitions for Pareto and prioritization to capture the semantics of "equally good" without violating the SPO property is a non-trivial issue. The *substitutable values* (SV) semantics [10] provides the most general way to achieve this goal by characterizing, which domain values can be considered as "equally good", hence being "substitutable". Using *trivial* SV-semantics only *equal* values are considered as equally good (Definition 4 and Definition 5 are examples for trivial SV-semantics), whereas for *regular* SV-semantics values that are *indifferent* are considered as equally good. However, regular SV-semantics is only valid for WOPs, otherwise SPO is violated.

Another form of preference combination is by associating numerical scores to each individual preference and then applying a combining function to compute a single score, which decides the "better-than" relationship. Such *weighted importance* between preferences is realized by the numerical ranking preference, cf. [8, 10].

Definition 6 (Numerical Ranking Preference): Given weak order preferences $P_1, \ldots, P_m$ with scoring functions $f_1, \ldots, f_m$. Then the numerical ranking preference $\text{RANK}_{F,d}$ ($d > 0$) with an m-ary combining function $F : \mathbb{R}^m \to \mathbb{N}_0$ is defined as:

$$(x_1, \ldots, x_m) <_P (y_1, \ldots, y_m) \iff \left\lceil \frac{F(f_1(x_1), \ldots, f_m(x_m))}{d} \right\rceil > \left\lceil \frac{F(f_1(y_1), \ldots, f_m(y_m))}{d} \right\rceil$$

Note that $\text{RANK}_{F,d}$ yields a WOP again. On the other hand Pareto and prioritization are not WOP preserving.

2.2 The BMO Query Model

The declarative query model for relational databases relies on relational calculus, which is equivalent to relational algebra. Queries under this model implement the semantics of hard constraints. All retrieved elements exactly satisfy the specified constraints. Therefore this model is prone to the empty-result effect mentioned earlier. When trying to overcome it by applying tedious parametric search, Boolean expert search or similar ad-hoc remedies, often the opposite effect of flooding the user with too many, mostly irrelevant results occurs. All of this can be avoided by supporting the BMO query model for preferences as follows.

Definition 7 (Preference Selection, BMO-set): Let $P = (A, <_P)$ be a preference and $R(A_1, \cdots, A_m)$ be a database relation with $A \subseteq \{A_1, \cdots, A_m\}$. Then the preference selection operator is defined by:

$$\sigma[P](R) := \{t \in R \mid \neg \exists t' \in R : t[A] <_P t'[A]\}$$

The result of preference selection (also named "winnow"-operator in [4]) is called BMO-set.

Preference selection is defined by means of negations, hence it defines a form of non-monotonic reasoning. It offers a cooperative query answering behavior by automatic matchmaking: The BMO query result adapts to the quality of the data in the database, defeating the empty result effect and reducing the flooding effect by filtering off worse results. Thus preferences are treated as *soft* constraints. Note that Pareto preference queries form a superset of the skyline operator [2]. However, the skyline operator only allows LOWEST and HIGHEST preferences. In contrast, a Pareto preference can be constructed from any preferences presented in Section 2.

3 The Preference SQL Query Language

Before illustrating the syntax of Preference SQL we shortly address the formal semantics of Preference SQL, which extends standard SQL with its exact match semantics by SPO preferences under the BMO query model. The declarative reasoning paradigm of SQL for relational databases is founded on the familiar model theory based on relational calculus, its operational equivalent being relational algebra. It is known that this can be

extended towards recursive relational databases. For preference queries with BMO semantics, subsumption models [13] can serve as a theoretical foundation for the declarative semantics, with an equivalent operational semantics expressed by the preference selection operator. Our current version of Preference SQL is an extension to SQL-92 compliant (i.e., non-recursive) relational databases.

Syntactically, Preference SQL [12, 6] extends the SELECT statement of SQL by an optional PREFERRING clause. A preference query selects all interesting tuples, i.e., tuples that are not dominated by other tuples. Preference SQL currently supports most of the SQL-92 standard as well as all previously mentioned base preferences together with the constructors for Pareto, prioritization and Rank. A Preference SQL query block has the following schematic design:

```
SELECT          ...  <selection>
FROM            ...  <table_reference>
WHERE           ...  <hard_conditions>
 PREFERRING     ...  <soft_conditions>
 GROUPING       ...  <attribute_list>
 BUT ONLY       ...  <but_only_condition>
GROUP BY        ...  <attribute_list>
HAVING          ...  <hard_conditions>
ORDER BY        ...  <attribute_list>
```

Figure 2: Preference SQL query block.

The keywords SELECT, FROM, WHERE, GROUP BY, HAVING, and ORDER BY are treated as the standard SQL query keywords. The other keywords are explained subsequently. For further syntax details of Preference SQL we refer to [1] and [12].

▷ `PREFERRING:` The PREFERRING clause specifies a (possibly very complex) preference P by means of the preference constructors stated in Section 2. P is evaluated on the result T of the hard conditions stated in the WHERE clause, returning the BMO-set of P. Note that in this way Preference SQL implements a generalization of **constrained skyline queries** [16, 5]. Empty result sets can only occur when the WHERE clause returns an empty result T. Note that all WOPs can be specified with either *trivial* or *regular* SV-semantics omitting or using the keyword REGULAR after the preference definition.

▷ `GROUPING:` Instead of evaluating P on the entire intermediate relation T, the GROUPING clause partitions T w.r.t. the given attribute list. Then P is evaluated on each partition of T separately.

▷ `BUT ONLY:` There are cases, where the BMO-set returned by the PREFERRING / GROUPING clause is too large for a given application scenario. Note that his tends to hold in particular for high-dimensional Pareto / skyline queries. Then within the BUT ONLY clause one can specifiy hard conditions similar to the WHERE clause. BUT ONLY is evaluated on the result of PREFERRING / GROUPING, hence implementing a generalization of **skyline queries with constraints**, cp. [16].

Example 1: We want to find all cars with their owners and address who live in Augsburg or Munich. The average fuel consumption on highway and city driving must be equal or less than 6 liter / 100 km. The cars should have a mileage between 40.000 km and 50.000 km, and a price around 12.000 Euro. A difference of 2.000 Euro does not matter. This Pareto preference is more important than the make being a BMW or an Audi. The full-text description of the car should contain “ABS brake”. This is equally important to the make. The preference is combined with a RANK preference on the age and the price of the car. Furthermore, only cars with a horsepower less than 100 hp should be presented after preference evaluation (BUT ONLY). The result has to be ordered by the registration date of the cars. This Preference SQL query is depicted in Figure 3.

```
SELECT o.id, a.id, c.id
FROM owner o, address a, car c
WHERE   a.city = 'Augsburg' OR a.city = 'Munich' AND
        a.id = o.idaddress AND o.id = c.idowner  AND
        (c.city_consumption + c.highway_consumption) / 2 <= 6
 PREFERRING  (c.mileage BETWEEN (40000 AND 50000) AND
                c.price AROUND 12000, 2000)                          PRIOR TO
             (c.price HIGHEST AND c.make IN ('BMW', 'Audi') REGULAR  AND
                c.description CONTAINS 'ABS brake'                   AND
             (c.age SCORE 'age_function', c.price SCORE 'price_function')
                RANK age_price_function )
 BUT ONLY c.hp < 100 ORDER BY c.reg_date;
```

Figure 3: A sample complex Preference SQL query.

The WHERE clause states the join conditions. In Preference SQL syntax, Pareto preferences are stated as "AND" in the PREFERRING clause, while prioritization is expressed by the keyword "PRIOR TO". The keywords "IN" represents a POS preference, "CONTAINS" expresses the full-text search. The RANK preference consists of score functions which must be implemented in Preference SQL itself. After the evaluation of the preferences the BUT ONLY clause filters out all result tuples which do not fulfill the horsepower condition.

Note that it is possible to specify several preferences on the same attribute. In the query above there are three preferences on the price of the car, which might for example come from different persons. We call such preferences *social group preferences* because one important application is to use them to model group decisions and group formations depending on individual group member preferences.

4 Preference Query Evaluation

4.1 The Preference SQL System Architecture

Extending existing SQL database systems towards Preference SQL requires some crucial design decisions. The current University version of Preference SQL adopts a (Java 1.6-based) middleware approach as depicted in Figure 4.

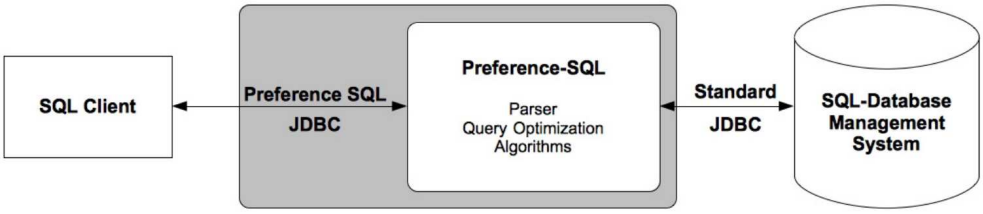

Figure 4: System architecture of Preference SQL

For a seamless application integration we have extended standard JDBC technology towards a Preference SQL / JDBC package. This enables the client application to submit Preference SQL queries through familiar SQL clients. The Preference SQL middleware parses the query and performs preference query optimization as described subsequently. Those parts of an optimized query execution plan, which directly correspond to SQL-92, are retranslated into SQL-92 and handed over to the attached SQL database system for evaluation. The new preference selection operator together with novel algebraic and cost-based query optimization methods are dealt with in the Preference SQL middleware. This approach has proven its flexibility and performance in various prototype applications conducted so far, see [11] for example.

4.2 Preference Query Optimization

Being an extension of standard SQL, Preference SQL can inherit the accumulated vast query optimization knowhow from relational databases. Moreover, additional optimization techniques are required to cope with the challenges posted by the preference selection operator for complex strict partial order preferences. Since Preference SQL always works with a standard SQL database system as back-end, another design goal for the preference query optimizer has been to offload parts of the query evaluation work to this back-end. Basically, in its current version a Preference SQL query is processed as follows:

1) The query gets parsed and transformed into an initial operator tree for preference relational algebra, which comprises classical relational algebra plus the preference selection operator.
2) Then heuristics for algebraic operator tree transformation are applied. For this purpose, the familiar principles known from standard relational databases had to be augmented by new transformation laws for preference relational algebra (see [6, 4, 5] for details).
3) For efficient evaluation of the preference selection operator in our middleware, special efficient algorithms had to be implemented. For instance, for Pareto / skyline queries this repertoire includes the Hexagon algorithm [14] and LESS [8]. The algorithm selection is guided by a cost-based query execution model.
4) Likewise guided by a cost-based model, the Preference query optimizer determines suitable sub-trees of the final optimized operator tree for offloading them to the back-end SQL system. Those sub-trees are re-translated into SQL and sent via JDBC to the attached back-end afterwards.
5) The entire result of the Preference SQL query is assembled in our middleware and returned to the requesting client by Preference SQL JDBC.

It has to be pointed out that many optimization methods applied above would become invalid if transitivity of preferences would get violated.

5 Summary and Outlook

We have presented an overview on the Preference SQL system, which extends standard SQL by preferences. Preference evaluation follows the BMO paradigm, treating preferences as soft constraints. For ease of use a variety of preference constructors on categorical as well as on numerical domains are available. Since preferences are always strict partial orders, by means of composition complex preferences can be formulated inductively. This includes Pareto preferences, prioritization and also numerical ranking. Seamless application integration is supported by a Preference SQL extension to JDBC, enabling easy access to standard SQL systems like, e.g., Oracle, MySQL or PostgresSQL. By this means, standard SQL clients like e.g. Squirrel or DBVisualizer, or even mobile iPhone or Android clients, can execute Preference SQL queries immediately. Our preference query optimization implements algebraic methods as well as cost-based approaches, including specialized evaluation methods like, e.g., the Hexagon algorithm for Pareto queries. A demo of Preference SQL is available at [1].

An enhancement is concerning a new Top-k interface for preference queries on top of the already established BMO evaluation. Also we are planning to advance existing text search capabilities to a more powerful preference based full-text search. Finally, the use of ontologies in combination with preferences is of interest. Besides these changes to the core functionality, current research efforts target the special support of Preference SQL for innovative application areas such as location-based services, mobile geo-services and social networking sites. Hence, the development of preference constructors on geospatial data that is e.g. stored in a PostGIS system as well as possibilities to extend context-awareness are ongoing research. Personalized routing problems will also ask for extensions towards recursive Preference SQL. In social networks, aspects of social matching, social group preferences and group recommendations are driving further developments of Preference SQL.

References

[1] The Preference SQL Syntax and Trial Version. *http://www.trial.preferencesql.com/*

[2] S. Börzsönyi, D. Kossmann, and K. Stocker. The Skyline Operator. In Proceedings of the 17th International Conference on Data Engineering (ICDE), pp. 421–430, Washington, USA, 2001.

[3] R. I. Brafman, and C. Domshlak. Preference Handling: An Introductory Tutorial. In AI Magazine 30 (1), pp. 58 - 86, Menlo Park, CA, USA, 2009.

[4] J. Chomicki. Preference Formulas in Relational Queries. In ACM Transactions on Database Systems (TODS), volume 28, pp. 427–466, New York, NY, USA, 2003.

[5] M. Endres and W. Kießling. Semi-Skyline Optimization of Constrained Skyline Queries. In Proceedings of the 22nd Australasian Database Conference (ADC), Perth, Australia, 2011.

[6] B. Hafenrichter and W. Kießling. Optimization of Relational Preference Queries. In Proceedings of the 16th Australasian Database Conference (ADC), pp. 175–184, Darlinghurst, Australia, 2005.

[7] S. Holland, and W. Kießling. Situated Preferences and Preference Repositories for Personalized Database Applications In Proceedings of the 23rd International Conference on Conceptual Modeling (ER), pp. 511–523, Shanghai, China, 2004.

[8] P. Godfrey, R. Shipley, and J. Gryz. Maximal Vector Computation in Large Data Sets. In Proceedings of the 31st International Conference on Very Large Data Bases (VLDB), pp. 229–240, Trondheim, Norway, 2005.

[9] W. Kießling. Foundations of Preferences in Database Systems. In Proceedings of the 28th International Conference on Very Large Data Bases (VLDB), pp. 311–322, Hong Kong, China, 2002.

[10] W. Kießling. Preference Queries with SV-Semantics. In Proceedings of the 11th International Conference on Management of Data (COMAD), pp. 15–26, Goa, India, 2005.

[11] W. Kießling, S. Fischer, and S. Döring. COSIMA^{B2B} - Sales Automation for E-Procurement. In The 6th IEEE Conference on E-Commerce Technology (CEC), pp. 59–68, Los Alamitos, CA, USA, 2004.

[12] W. Kießling and G. Köstler. Preference SQL - Design, Implementation, Experiences. In Proceedings of the 28th International Conference on Very Large Data Bases (VLDB), pp. 990–1001, Hong Kong, China, 2002.

[13] G. Köstler, W. Kießling, H. Thöne, and U. Güntzer. Fixpoint Iteration with Subsumption in Deductive Databases. In Journal of Intelligent Information Systems, Vol. 4, pp. 123–148, 1995.

[14] T. Preisinger and W. Kießling. The Hexagon Algorithm for Evaluating Pareto Preference Queries. In Proceedings of the 3rd Multidisciplinary Workshop on Advances in Preference Handling (M-PREF in conjunction with VLDB '07), Vienna, Austria, 2007.

[15] K. Stefanidis, G. Koutrika, and E. Pitoura. A Survey on Representation, Composition and Application of Preferences in Database Systems. In ACM Transactions on Database Systems (TODS), To appear in Vol. 36, Issue 4, New York, NY, USA, 2011.

[16] M. Zhang, and R. Alhaji. Skyline queries with constraints: Integrating skyline and traditional query operators. In Data & Knowledge Engineering, pp. 153–168, Elsevier, 2010.

Contextual Database Preferences

Evaggelia Pitoura
Dept. of Computer Science
University of Ioannina, Greece
pitoura@cs.uoi.gr

Kostas Stefanidis
Dept. of Computer Science and Engineering
Chinese University of Hong Kong, Hong Kong
kstef@cse.cuhk.edu.hk

Panos Vassiliadis
Dept. of Computer Science
University of Ioannina, Greece
pvassil@cs.uoi.gr

Abstract

As both the volume of data and the diversity of users accessing them increase, user preferences offer a useful means towards improving the relevance of the query results to the information needs of the specific user posing the query. In this article, we focus on enhancing preferences with context. Context may express conditions on situations external to the database or related to the data stored in the database. We outline models for expressing both types of preferences. Then, given a user query and its surrounding context, we consider the problem of selecting related preferences to personalize the query.

1 Introduction

Personalization in databases aims at addressing the explosion of the amount of data currently available to an increasingly wider spectrum of users by presenting to users only those items that are of interest to them. Preferences have been used as a means to address this challenge through supporting the expression of user interests, likes and dislikes [15]. Most often preferences depend on the surrounding context. For instance, the choice of which movie to see or which place to visit may depend on the current weather, location or the people accompanying the user. In this respect, user preferences are *context-dependent*.

Context is a general term used in several domains [3, 4, 7]. We differentiate between two general context types: (i) internal and (ii) external context. *Internal context* captures conditions that involve the data items stored in the database for which preferences are expressed. *External context* involves conditions outside the database. Common types of external context include the computing context (e.g., network connectivity, nearby resources), the user context (e.g., profile, location), the physical context (e.g., noise levels, temperature) and time [5].

In this paper, we focus on specifying contextual preferences and on selecting appropriate preferences for personalizing a user query. In general, a contextual preference specification has two parts: a preference part that specifies the preference and a context part that specifies the conditions under which the given preference holds. We present a multidimensional model for expressing conditions regarding the external context that allows the

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

	mid	*Title*	*Year*	*Director*	*Genre*	*Language*	*Duration*
t_1	m_1	Casablanca	1942	M. Curtiz	drama	English	102
t_2	m_2	Psycho	1960	A. Hitchcock	horror	English	109
t_3	m_3	Happy-Go-Lucky	2006	M. Leigh	comedy	English	118

Figure 1: Example database relation.

expression of contextual preferences at various level of detail, for example both at the level of a city or a country [16, 17]. We also consider the preference selection problem, that is, given a set of contextual preferences and a query, determining which preferences are the most relevant to the query. We focus on context aspects and consider as relevant those preferences whose context is more general of that of the query. We call this problem *context resolution*. Finally, we present two data structures, *the preference graph* and the *profile tree*, to support efficient context resolution.

2 A Contextual Preference Model

Preferences express user interests, likes or dislikes. In its most general form, a contextual preference specification $\mathcal{CP} = (\mathcal{C}, \mathcal{P})$ is a pair, where the context specification part $\mathcal{C}$ expresses the conditions under which the preference specified by $\mathcal{P}$ holds. In the following, we briefly review approaches to preference specification and then focus on the specification of a multidimensional model for context. As a running example, we shall use a single relation with schema: *Movies (mid, Title, Year, Director, Genre, Language, Duration)*. An example relation (instance) of *Movies* with three tuples is shown in Fig. 1.

Preference Specification. Preference can be specified either qualitatively or quantitatively. In the *qualitative approach*, a preference $\mathcal{P}$ is specified as a binary relation $r_{\mathcal{P}}$ over a set D of items of interest, i.e., $r_{\mathcal{P}} \subseteq D \times D$ [6]. For relational databases, many alternatives exists for the domain D over which preferences can be specified, thus allowing preferences with various granularities. Let us first assume that preferences are expressed over tuples of a single database relation R. Then, for two tuples $t_i, t_j \in R$, preference $(t_i, t_j) \in r_{\mathcal{P}}$, also denoted as $t_i \succ_{\mathcal{P}} t_j$, means that tuple t_i is preferred over tuple t_j. For example, for the database in Fig. 1, preference $t_1 \succ_{\mathcal{P}} t_2$ specifies that "Casablanca" is preferred over "Psycho". Preference relations can be also defined over attributes to indicate their importance or relevance. For instance, for our example database, preference *Director* $\succ_{\mathcal{P}}$ *Duration* means that the director of a movie is considered more important or more relevant than its duration. Preferences can be also defined at the level of attribute values, for instance *comedy* $\succ_{\mathcal{P}}$ *drama* indicates a preference of comedies over dramas.

Qualitative preferences can be also specified by using conditions: for two items $d_i, d_j \in D$, $d_i \succ_{\mathcal{P}} d_j$, if $Cond(d_i, d_j)$, where *Cond* is a condition that must be satisfied for the preference to hold. For example, preference $t_i \succ_{\mathcal{P}} t_j$, if $t_i.Year > t_j.Year$, expresses a preference for recent movies. We can also specify preferences that involve tuples from more than one relation as well as conditions *Cond* whose evaluation is not based solely on the values of the items involved in the preference. The latter are called *extrinsic preferences*. An example extrinsic preference would be that a movie directed by M. Leigh is preferred over a movie directed by A. Hitchcock, if there exist no Spanish speaking movies.

In the *quantitative* approach, a preference $\mathcal{P}$ is specified through a preference function $f_{\mathcal{P}}: D \rightarrow [0,1]$ that assigns to each item $d \in D$ a preference degree or score $f_{\mathcal{P}}(d)$ usually in [0, 1]. Larger scores indicate higher degrees of interest, i.e., for $d_i, d_j \in D$, $f_{\mathcal{P}}(d_i) > f_{\mathcal{P}}(d_j) \Rightarrow d_i \succ_{\mathcal{P}} d_j$. Again, there are many alternatives regarding the domain D over which a preference function is defined. For example, quantitative preferences defined over join conditions between two relations have been used to express the strength of the relatedness between the two relations [12]. Function $f_{\mathcal{P}}$ may be also defined using conditions, i.e., $f_{\mathcal{P}}(d) = c$, if $Cond(d)$, where $d \in D$, $c \in$ [0, 1] and *Cond* is a condition. For example, preference $f_{\mathcal{P}}(t) = 0.9$, if $t.Genre$ = *drama* defined over tuples of

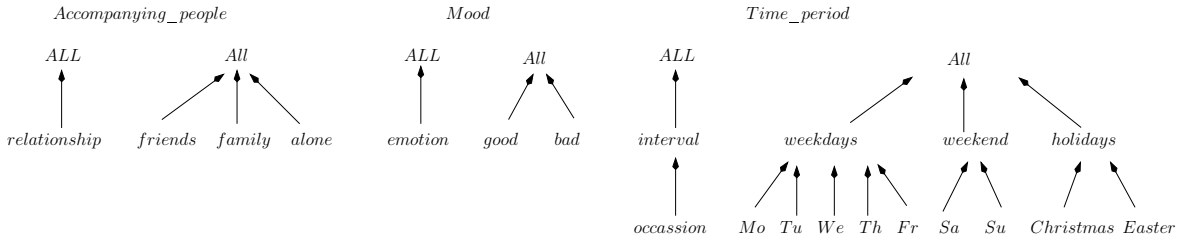

Figure 2: Hierarchy schema and concept hierarchy of *Accompanying_people*, *Mood* and *Time_period*.

the Movie relation assigns score 0.9 to all dramas.

A Multidimensional Context Model. We make a general distinction between internal and external context. *Internal* context refers to conditions that involve data items stored in the database, for example, the fact that a preference for a specific director is applicable under the condition that the genre of the movie is *drama*. *External* context expresses conditions that involve information outside the database. Note that in the case of preferences specified through conditions, the *Cond* part may be considered as a specification of internal context. In the rest of this section, we focus on external context and introduce a multidimensional model for capturing it.

A variety of models for (external) context have been proposed (see for example [20] for a survey). We follow a data-centric approach. We call *context environment*, CE, a set of *n context parameters*: $CE = \{C_1, \ldots, C_n\}$, where each context parameter C_i, $1 \leq i \leq n$, captures information that is not part of the database, such as the *user location* or the current *weather*. For our movie database, let us assume that the context environment consists of context parameters *Accompanying_people*, *Mood* and *Time_period*. Each context parameter takes values from a hierarchical domain, so that different levels of abstraction for the captured context data are introduced.

In particular, each context parameter has multiple levels organized in a hierarchy schema. Let C be a context parameter with $m > 1$ levels, L_i, $1 \leq i \leq m$. We denote its hierarchy schema as $L = (L_1, \ldots, L_m)$. L_1 is called the lowest or most detailed level of the hierarchy schema and L_m the top or most general one. We define a total order among the levels of L such that $L_1 \prec \ldots \prec L_m$ and use the notation $L_i \preceq L_j$ between two levels to mean $L_i \prec L_j$ or $L_i = L_j$. Fig. 2 depicts the hierarchy schemas of the context parameters of our running example. For instance, the hierarchy schema of context parameter *Time_period* has three levels: *occasion* (L_1), *interval* (L_2) and the top level *ALL* (L_3). Each level L_j, $1 \leq j \leq m$, is associated with a domain of values, denoted $dom_{L_j}(C)$. For all parameters, their top level has a single value *All*, i.e., $dom_{L_m}(C) = \{All\}$. A *concept hierarchy* is an instance of a hierarchy schema, where the concept hierarchy of a context parameter C with m levels is represented by a tree with m levels with nodes at each level j, $1 \leq j \leq m$, representing values in $dom_{L_j}(C)$. The root node (i.e., level m) represents the value All. Fig. 2 depicts the concept hierarchies of the context parameters of our running example. For instance, for the context parameter *Time_period*, *holidays* is a value of level *interval*. The relationship between the values at the different levels of a concept hierarchy is achieved through the use of a family of ancestor and descendant functions [22]. For example, for the concept hierarchies in Fig. 2, $anc_{L_1}^{L_2}(Christmas) = holidays$ whereas $desc_{L_1}^{L_2}(weekend) = \{Sa, Su\}$. Finally, we define the domain, $dom(C)$, of C as: $dom(C) = \cup_{j=1}^{m} dom_{L_j}(C)$.

A *context state cs* of a context environment $CE = \{C_1, \ldots C_n\}$ is an n-tuple $(c_1, \ldots, c_n)$, where $c_i \in dom(C_i)$, $1 \leq i \leq n$. For instance, (*friends*, *good*, *holidays*) and (*friends*, *All*, *Christmas*) are context states for our movie example. The set of all possible context states, called *world CW*, is the Cartesian product of the domains of the context parameters: $CW = dom(C_1) \times \cdots \times dom(C_n)$.

The context specification part $\mathcal{C}$ of a contextual preference $\mathcal{CP} = (\mathcal{C}, \mathcal{P})$ specifies a set of context states or situations in which $\mathcal{P}$ holds. $\mathcal{C}$ is a multi-parameter context descriptor defined as follows. A *single parameter context descriptor* $scod(C)$ of a context parameter C is an expression of the form $C \in \{v_1, \ldots, v_l\}$, where $v_k \in dom(C)$, $1 \leq k \leq l$. We use the notation $Context(scod(C_i)) = \{v_1, \ldots, v_l\}$. For example, for *Time_period*, a single parameter context descriptor is $Time\_period \in \{Fr, weekend\}$. A multi-parameter context descriptor *cod*

is an expression of the form $\wedge_{j=0}^{k}$ $scod(C_{i_j})$, $1 \leq k \leq n$, where $i_j \in \{1, \ldots, n\}$, $scod(C_{i_j})$ is a single context parameter descriptor for C_{i_j} and there is at most one single parameter context descriptor for each C_{i_j}. A multi-parameter context descriptor specifies a set of context states computed by taking the Cartesian product of the contexts of all the single parameter context descriptors that appear in the descriptor. If a multi-parameter context descriptor does not contain descriptors for a context parameter, we assume that the values of the absent context parameter are irrelevant. In particular, if a context parameter C_i is missing from a multi-parameter context descriptor, we assume the default condition $C_i \in \{All\}$. For example, the multi-parameter context descriptor cod = (*Accompanying_people* $\in$ {*friends*, *family*} $\wedge$ *Time_period* $\in$ {*holidays*}) defines the following two context states: $Context(cod)$ = {(*friends*, *All*, *holidays*), (*family*, *All*, *holidays*)}.

We call *profile* the set of all contextual preferences $(cod_i, \mathcal{P}_i)$, available to an application. The context $Context(Pr)$ of a profile Pr is the union of the contexts of all context descriptors that appear in Pr, that is, $Context(Pr) = \cup_i Context(cod_i)$, where $(cod_i, \mathcal{P}_i) \in P$.

3 Contextual Preference Selection

Query personalization refer to the process of using preferences to adapt the results of a query based on the interests of users as expressed through their preferences. Adaptation may be achieved through ranking the results, selecting representative subsets of the results or using different visual or otherwise presentations of the results. Preferences may be employed to personalize the query results either in a preprocessing or in a postprocessing step. As a preprocessing step, user preferences are used to rewrite or augment the original query, for example, by adding selection conditions to the query to incorporate preferences. Such selection conditions are in general considered soft constraints as opposed to the hard constraints expressed by the selection conditions present at the original query [10]. As a postprocessing step, user preferences are used after the execution of the original query to personalize its results, for example by re-ranking or filtering them.

Irrespectively of when preferences are used during query processing, a common issue is which of the preferences in the user profile to use for personalizing each specific query. The number of preferences to be used is central for the success of personalization, since selecting too many preferences may result to over-specialization, while selecting too few preferences may not be effective. Preference selection is based on the relevance between the preference and the query. In the following, we focus on defining formally the relevance between the context of a preference and the context of a query.

Let q be a query. We use cod^q to denote the multi-parameter context descriptor that specifies the context associated with q: $Context(q) = Context(cod^q)$. Such context descriptors may be postulated by the application or be explicitly provided by the users as part of their queries. Typically, in the first case, the context implicitly associated with a query corresponds to the current context, that is, the context surrounding the user at the time of the submission of the query. Besides this implicit context, we also envision queries that are explicitly augmented with multi-parameter context descriptors by the users issuing them. For example, a user may pose an exploratory query asking for a movie to see with *friends* during *Christmas*. The context associated with a query may correspond to a single context state, where each context parameter takes a specific value from its most detailed domain. In other cases, it may be possible to specify the query context using only rough values, for example, when context values are provided by sensor devices with limited accuracy. In such cases, a context parameter may take a single value from a higher level of the hierarchy or even more than one value.

Let us first consider a simple example where the context descriptor of q is (*Accompanying_people* $\in$ {*friends*} $\wedge$ *Mood* $\in$ {*good*} $\wedge$ *Time_period* $\in$ {*Christmas*}) that specifies a single context state cs^q = (*friends*, *good*, *Christmas*) as the context of q. If there exist a preference in the profile whose context includes a context state cs such that $cs = cs^q$, called an *exact match*, then this is a relevant preference that can be used to personalize q. Assume now that there is no exact match for cs^q in $Context(Pr)$. For example, assume that Pr includes just the first three preference specifications of Fig. 3(a). To simplify presentation, in the following, we consider

preferences with a single context state. Intuitively, in the absence of an exact match, we would like to use those preferences in Pr whose context "covers" that of the query. Formally, a context state $cs^1 = (c_1^1, \ldots, c_n^1) \in CW$ *covers* a context state $cs^2 = (c_1^2, \ldots, c_n^2) \in CW$ if $\forall\, k$, $1 \leq k \leq n$, $c_k^1 = c_k^2$ or $c_k^1 = anc_{L_i}^{L_j}(c_k^2)$ for some levels $L_i \prec L_j$. In our example, the context states of p_1 and p_2 cover c^q, whereas that of p_3 does not. It can be shown that the cover relation imposes a partial order among context states.

Although the context states of both p_1 and p_2 cover c^q, the context state of p_1 is more closely related to c^q. This is formalized through the notion of the most specific state or tight cover. Given a profile Pr and a context state cs^1, we say that a context state $cs^2 \in Context(Pr)$ is a *tight cover* of cs^1 in Pr, if and only if:

(i) cs^2 covers cs^1 and

(ii) $\neg\exists\, cs^3 \in Context(Pr)$, $cs^3 \neq cs^2$, such that cs^2 *covers* cs^3 and cs^3 *covers* cs^1.

We now provide a formal definition of context resolution, that is, of the process of selecting appropriate preferences from a profile based on context.

Definition 1 (Context Resolution Set): Given a profile Pr and a contextual query q, a set RS of context states, $RS \subseteq Context(Pr)$, is called a context resolution set for q if (i) for each context state $cs^q \in Context(q)$, there exists at least one context state cs in RS such that cs is a tight cover of cs^q in Pr and (ii) cs belongs to RS only if there is a $cs^q \in Context(q)$ for which cs is a tight cover in Pr.

Note that there may be more than one tight cover of a query context state. For example, consider again query context $c^q = (friends, good, Christmas)$ and the first four preference specifications in Fig 3(a). Both the context states of p_1 and p_4 are tight covers of c^q. For a set of context states to qualify as a context resolution set, it must include *at least one* of them. In [17], we provide a systematic way of ranking context states based on their distances from the state of the query. The definition of distance between two context states is based on the path distance between their values in the corresponding concept hierarchy and on the relative size of their domains. Distances can be used to select *exactly one*, i.e., the most similar, tight cover of each query state, thus, creating the *smallest* context resolution sets. They can also be used to include in the context resolution set more than one tight cover per query context state, for example, by selecting among the tight covers of a query context state, the k ($k > 1$) most similar to it. This provides a means for controlling the degree of personalization. Finally, note that for a query q and profile Pr, there may be no tight cover and thus no context resolution set. In this case, we can either execute q as a regular query, i.e., without using any preference, or relax our requirement for relevance and consider context states that are similar to the query state although not necessarily tight covers of it [19]. Our usability studies have indicated that in most cases using exactly one tight cover produces slightly more satisfying results than using more than one tight cover, whereas using preferences with relaxed context states is better that using none.

4 Indexes for Contextual Preferences

In this section, we focus on the efficient computation of context resolution sets. One way to achieve this is by sequentially scanning all context states of all preferences in Pr. To improve response time and storage overheads, we consider indexing the preferences in Pr based on the context states in $Context(Pr)$. To this end, we introduce two data structures: the *preference graph* and the *profile tree*. The preference graph exploits the cover relation between context states.

Definition 2 (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.

For example, Fig. 3(b) depicts the preference graph for the preferences in Fig. 3(a). Note that, when there is at least one preference with context state $(All, \ldots, All)$, the graph has a single root. The preference graph is

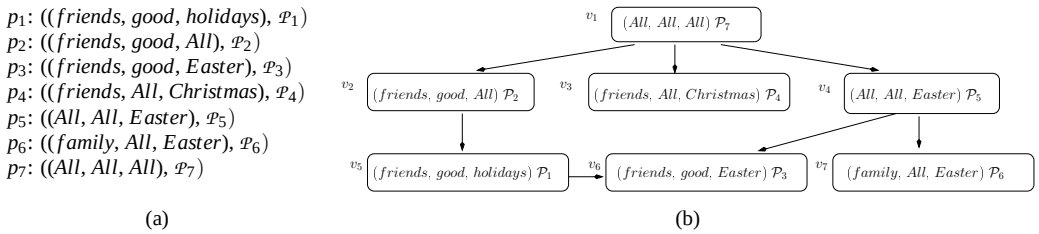

Figure 3: (a) Example set of preferences and (b) the corresponding preference graph.

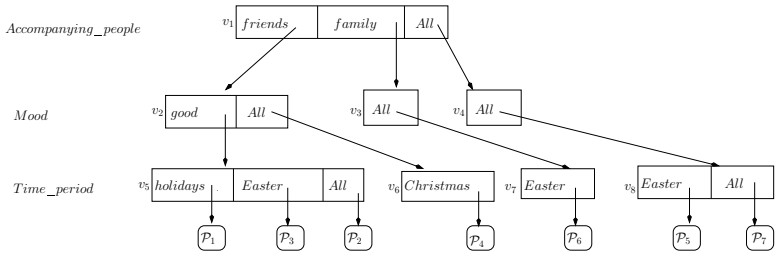

Figure 4: The profile tree for the example preferences of Fig. 3(a).

acyclic, since the cover relation over context states is a partial order. The size of PG_{Pr} depends on the number of distinct context states in Pr.

Given a context state cs and a profile Pr, the context states in $Context(Pr)$ that are tight covers of cs are located by a top-down traversal of the preference graph PG_{Pr} starting from the nodes in V_{Pr} with no incoming edges. Search stops at a node v of the graph, if v is a leaf node or the context state of v does not cover cs. The context state of a node is included in the result if (i) it is a leaf node whose context state covers cs or (ii) the context states of all of its children do not cover cs. For example, consider the preference graph in Fig. 3(b) and input context state cs = (*friends*, *good*, *Christmas*). Search starts at the root node v_1 and since its context state covers cs, it proceeds to its children. Search stops at v_3 which is a leaf node and at v_4 whose context state does not cover cs. The context state (*friends*, *All*, *Christmas*) of v_3 is returned, since v_3 is a leaf node whose context state covers cs. From v_2, search proceeds to node v_5 and then to leaf node v_6. The context state (*friends*, *good*, *holidays*) of v_5 is returned, since the context state of its only child does not cover cs.

The profile tree explores common prefixes of context states in the profile. We say that a value $c \in dom(C_i)$ *appears* in a context state $cs = (c_1, \ldots, c_i, \ldots, c_n)$, if $c_i = c$. The *length k prefix* of $(c_1, \ldots, c_k \ldots, c_n)$ is $(c_1, \ldots, c_k)$. A profile tree has n+1 levels. Each one of the first n levels corresponds to one of the context parameters. We use C_{t_i} to denote the parameter mapped to level i, $t_i \in \{1, \ldots, n\}$. The last level, level $n+1$, includes the leaf nodes.

Definition 3 (Profile Tree): The profile tree T_{Pr} of a profile Pr is a tree with n+1 levels constructed as follows.

(i) Each internal node at level k, $1 \leq k \leq n$, contains a set of cells of the form $[val, pt]$ where $val \in dom(C_{t_k})$ and pt is a pointer to a node at the next tree level, i.e., level $k+1$.

(ii) Each leaf node at level $n+1$ contains one or more preference.

(iii) At the first level of the tree, there is a single root node that contains a $[c, p]$ cell for each value $c \in dom(C_{t_1})$ that appears in a context state $cs \in Context(Pr)$.

(iii) At level k, $1 < \mathrm{k} \leq n$, there is one node, say node v_o, for each $[c_o, p_o]$ cell of each node at level $k-1$. Node v_o includes a $[c, p]$ entry for each value $c \in C_{t_k}$ that appears in a context state cs such that $cs \in Context(Pr)$ and c_o also appears in cs. The corresponding pointer p_o points to v_o.

(iv) There is a leaf node, say node v_l for each $[c, p]$ cell of a node at level n. Pointer p points to this leaf node. Let $cs = (c_{t_1}, \ldots, c_{t_n})$ be the context state formed by the values of the cells on the path from the root node to v_l. The leaf node v_l contains the preferences associated with context state $cs = (c_1, \ldots, c_n)$.

For example, for the preferences in Fig. 3(a), the profile tree depicted in Fig. 4 is constructed. Note that there is exactly one root-to-leaf path for each context state *cs* in $Context(Pr)$. Each leaf node maintains the preferences associated with the corresponding context state. The size of the profile tree T_{Pr} depends on the number of common prefixes of the context states in $Context(Pr)$ and on the assignments of context parameters to tree levels.

The profile tree T_{Pr} supports the efficient look-up of a context state $cs = (c_1, \ldots, c_n)$, since at each level i we just need to follow the pointer of the cell with value c_{t_i}. If *cs* exists in *Pr*, then a single path is followed. If an exact match for *cs* does not exist in *Pr*, the context states in $Context(Pr)$ that cover *cs* are located by a top-down, breadth-first traversal of T_{Pr}. These context states need to be processed further to identify which of them are tight covers. In particular, at each level i of the tree, all paths of length i whose context state is either the same or covers the prefix $(c_{t_1}, \ldots, c_{t_i})$ of the input context state *cs* are maintained as candidates. For example, for the profile tree of Fig. 4 and input context state *cs* = (*friends*, *good*, *Christmas*), search starts from the root node and follows the pointers of the cells with values *friends* and *All* (i.e., the same or ancestor values of *friends*) to nodes v_2 and v_4 respectively. At the next level (level two): (i) from node v_2, the pointers associated with value *good* and *All* is followed to nodes v_5 and v_6 respectively, and (ii) from node v_4, the pointer associated with value *All* is followed to node v_8. At the next level (level three): (i) from node v_5, the pointers associated with *holidays* and *All* (ii) from node v_6, the pointer associated with *Christmas* and (iii) from node v_8, the pointer associated with *All* are followed. Thus, preferences $\mathcal{P}_1$, $\mathcal{P}_2$, $\mathcal{P}_4$, and $\mathcal{P}_7$, i.e., all preferences whose context state covers that of *cs*, are returned.

The profile tree is in general smaller than the preference graph since it takes advantage of repetitions of prefixes of context states. With the profile tree, exact matches are resolved by a simple root-to-leaf traversal, while for non exact matches, multiple candidate paths are maintained. The preference graph performs similarly for both exact and non exact matches. Note that the profile tree returns covering context states, thus, to compute tight covers, the extra step of sorting these context states based on their distances to the query context state is required. Finaly, note that so far, we have used the preference graph and the profile tree to locate tight covers of a single context state. Algorithms for locating tight covers of more than one context state can be designed by representing the context states in $Context(q)$, using a preference graph or a profile tree.

5 Summary and Related Work

In this paper, we have briefly presented our work on contextual preferences [16, 17, 18]. Our focus is on annotating preferences with contextual information. Context is modeled using a set of context parameters that take values from hierarchical domains, thus, allowing different levels of abstraction for the captured context data. A context state corresponds to an assignment of values to each of the context parameters from its corresponding domain. Preferences are augmented with context descriptors that specify the context states in which a preference holds. Similarly, each query is related with a set of context states. We have considered the problem of identifying those preferences whose context states are the most similar to that of a given query. We called this problem *context resolution*. To realize context resolution, we have proposed two data structures, namely the preference graph and the profile tree, that allow for a compact representation of contextual preferences.

The research literature on preferences is extensive, see, for example, [15] for a recent survey on using preferences in database systems. In the *quantitative* approach (e.g., [2, 11]), preferences are expressed using scoring functions that assign degrees of interest to specific pieces of information. In the *qualitative* approach (for example, [6, 10, 8]), preferences between pieces of information are specified directly, typically using binary preference relations. Our model is orthogonal to the approach taken to represent preferences. Contextual preferences consider either internal [1, 14] or external context [21, 9, 13]. Work on internal context focuses mainly on

efficiently ranking database tuples using preferences. The main novelty of our model lies in multidimensionality that allows flexibility in expressing preferences.

References

[1] R. Agrawal, R. Rantzau, and E. Terzi. Context-sensitive ranking. In *SIGMOD*, 2006.

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

[3] C. Bolchini, C. Curino, G. Orsi, E. Quintarelli, R. Rossato, F. A. Schreiber, and L. Tanca. And what can context do for data? *Commun. ACM*, 52(11):136–140, 2009.

[4] P. Brézillon. Context in artificial intelligence: A survey of the literature. *Computers and Artificial Intelligence*, 18(4), 1999.

[5] G. Chen and D. Kotz. A survey of context-aware mobile computing research. Technical report, Dartmouth College, Hanover, NH, USA, 2000.

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

[7] A. K. Dey. Understanding and using context. *Personal and Ubiquitous Computing*, 5(1):4–7, 2001.

[8] P. Georgiadis, I. Kapantaidakis, V. Christophides, E. M. Nguer, and N. Spyratos. Efficient rewriting algorithms for preference queries. In *ICDE*, 2008.

[9] S. Holland and W. Kießling. Situated preferences and preference repositories for personalized database applications. In *ER*, 2004.

[10] W. Kießling. Foundations of preferences in database systems. In *VLDB*, 2002.

[11] G. Koutrika and Y. E. Ioannidis. Constrained optimalities in query personalization. In *SIGMOD*, 2005.

[12] G. Koutrika and Y. E. Ioannidis. Personalized queries under a generalized preference model. In *ICDE*, 2005.

[13] A. Miele, E. Quintarelli, and L. Tanca. A methodology for preference-based personalization of contextual data. In *EDBT*, 2009.

[14] K. Stefanidis, M. Drosou, and E. Pitoura. Perk: personalized keyword search in relational databases through preferences. In *EDBT*, 2010.

[15] K. Stefanidis, G. Koutrika, and E. Pitoura. A survey on representation, composition and application of preferences in database systems. *ACM Trans. Database Syst.*, 36(4):To appear, 2011.

[16] K. Stefanidis and E. Pitoura. Fast contextual preference scoring of database tuples. In *EDBT*, 2008.

[17] K. Stefanidis, E. Pitoura, and P. Vassiliadis. Adding context to preferences. In *ICDE*, 2007.

[18] K. Stefanidis, E. Pitoura, and P. Vassiliadis. A context-aware preference database system. *International Journal of Pervasive Computing and Communications*, 3(4):439–460, 2007.

[19] K. Stefanidis, E. Pitoura, and P. Vassiliadis. On relaxing contextual preference queries. In *MDM*, 2007.

[20] T. Strang and C. Linnhoff-Popien. A context modeling survey. In *Workshop on Advanced Context Modelling, Reasoning and Management*, 2004.

[21] A. H. van Bunningen, L. Feng, and P. M. G. Apers. A context-aware preference model for database querying in an ambient intelligent environment. In *DEXA*, 2006.

[22] P. Vassiliadis and S. Skiadopoulos. Modelling and optimisation issues for multidimensional databases. In *CAiSE*, 2000.

Personalized DBMS: an Elephant in Disguise or a Chameleon?

Georgia Koutrika
IBM Almaden Research Center, USA

Abstract

A query, expressed in a structured query language such as SQL, describes precisely the data that should comprise the answer and a non-empty answer is returned only if it satisfies all query criteria. However, the Boolean database query model has limitations. A user may run into empty answers when the query conditions are too restrictive or sift through too many results that match the query. User preferences naturally blend into queries allowing to explore alternatives or prioritize and filter available choices. In addition, they can be used to generate answers that are personalized to the user. In this paper, we explain the need towards personalized and preference-aware query answering in the context of databases and we discuss how it can be enabled. Our objective is to discuss the pros and cons of existing approaches and the challenges and opportunities towards a truly personalized preference-aware DBMS.

1 Introduction

Database systems answer queries accurately and deterministically. A query, expressed in a structured query language such as SQL, describes precisely the data that should comprise the answer and a non-empty answer is returned only if it satisfies all query criteria. The DBMS guarantees that the same exact answer will be generated every time the query is issued as long as the data does not change. This exact-match query paradigm serves well many scenarios. When querying your bank account balance, you always expect to get the exact amount and this amount should be the same every time a check balance request is issued as long as no deposits or withdrawals have taken place. Or when a credit card company is preparing customer bills, it is important that the correct amounts are computed and displayed on each bill.

However, the database query model may sometimes be very strict. For example, when accessing databases on the web, whose schema and contents are unknown, it may be difficult to formulate accurate queries. In this context, a user may run into either of two problems: (*i*) the empty-answer problem when the query conditions are too restrictive or (*ii*) the too-many-answers problem when too many results match the query. Furthermore, information abundance and user heterogeneity are additional factors that make the exact-match query paradigm often not appropriate. Due to information abundance, showing all possible information that matches a query may be overwhelming for the user. User heterogeneity means that different users may find different things relevant when searching because of different preferences and goals [17]. Especially on the web, it is critical to provide the 'right' answer to a user rather than the exact same answer to everyone for the same query.

Introducing preferences in query answering can help cope with the issues mentioned above. The empty-answer problem can be tackled by relaxing some of the hard constraints in the query, i.e., considering them as

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

Movies

movie_id	title	year	genre
m_1	Anger Management	2003	comedy
m_2	The Proposal	2009	comedy
m_3	The Odd couple	1968	comedy
m_4	New in Town	2009	comedy

Actors

actor_id	movie_id	name
a_1	m_1	Adam Sandler
a_2	m_1	Jack Nicholson
a_3	m_2	Sandra Bullock
a_4	m_3	Walter Matau
a_5	m_4	Renée Zellweger

Figure 1: An example database instance

soft or as user wishes or by replacing them by constraints that capture user preferences related to the query. The too-many-answers problem can be tackled by strengthening the query with additional conditions expressing user preferences to restrict the results. In both cases, results can be prioritized according to how well they match the preferences in the query.

Furthermore, incorporating user preferences in query answering can help generate answers customized to each user. To illustrate this personalized answer paradigm, let us consider a web-based movie database. Figure 1 shows a snapshot containing new arrivals. Two users, Bob and Diana, issue the following queries on: (Q_1) comedies released after 2000; (Q_2) movies with Ben Stiller; and (Q_3) new comedies. Based on the exact-match database query paradigm, the same answers will be returned to both users, which are the following: (*i*) $\{m_1, m_2, m_4\}$ for Q_1, (*ii*) $\{\}$ for Q_2 and (*iii*) all new arrivals for Q_3. On the other hand, a personalized database query model would possibly generate a different answer for each query depending on the targeted user. For instance, for Q_1, Bob could receive $\{m_1, m_2\}$ because he is not a fan of actress R. Zelweger. For Q_2, showing movies that are a close match to the movies described by the query would potentially be better than an empty answer. For example, for Diana who also likes Adam Sandler and has liked movies where both Ben Stiller and Adam Sandler have co-starred, m_1 might be suggested as an alternative answer. Finally, results for Q_3 can be ordered according to a user's preferences so that each individual can easily sift through them.

Preferences naturally relax the exact-match query model by allowing to prioritize and filter available choices (as in Q_1 and Q_3) or to explore alternatives (as in Q_2). Moreover, embedding preferences in queries enable a shift from 'consensus relevancy', where the computed relevancy for the entire population is presumed relevant for each user, towards 'personal relevancy', where relevancy is computed based on each individual's characteristics.

In this article, we review related work on *preference-aware query answering* in databases. We discuss (*i*) how preferences are represented in the context of database queries, (*ii*) how the answer of a query with preferences is defined, and (*iii*) how preference-aware query answering is implemented. This review aims at showing how tightly preferences are currently coupled with database queries and sharing our vision regarding opportunities and challenges in fully implementing preferences as first-class citizens inside the database engine by changing both the database query model and internal code. Finally, this article highlights the challenges in going from preference-aware query answering to *personalized preference-aware query answering*, where the database automatically customizes answers taking into account user preferences.

2 Preference Representations

Understanding preferences and finding appropriate representations for them is a real challenge. Preferences have been studied for several years in fields such as philosophy (e.g., [8]), psychology (e.g., [18]) and economics (e.g., [6]). In recent years, preference modeling has drawn the attention of researchers from computer science fields including artificial intelligence (e.g., [22]) and databases.

In databases, preference representation approaches (preference models) are divided into two categories. *Qualitative preferences* are specified using binary preference relations that relatively compare tuples. Preference relations may be specified using logical formulas [4] or special preference constructors [11]. *Quantitative preferences* are expressed by assigning scores to particular tuples [1] or query conditions that describe sets of

tuples (e.g., [14]), where a score expresses degree of interest.

To illustrate, let us consider the database schema of Figure 1 and assume that t denotes a tuple and $t[A]$ stands for the projection of t on one of its attributes, A. A preference for recent movies can be expressed in a qualitative way as follows: movie t_i is preferred over movie t_j iff $t_i[year] > t_j[year]$. This formulation is natural for humans and results in a partial ordering of the query results. Absolute specification of the significance of a preference is not possible. On the other hand, using a scoring function such as $f(t) = 0.001 \times t[year]$, the same preference can be captured in a quantitative way. A movie t_i is preferred over movie t_j if the score of t_i is higher than the score of t_j. This formulation leads to a total ordering of the results based on user preferences.

Existing works have studied various types of preferences following a qualitative or quantitative approach including likes and dislikes [11, 14], context-dependent preferences [29], set preferences [23], and multi-granular preferences [13]. A survey on preference models in the context of databases can be found in [19].

3 Integrating Preferences into Queries

In the exact-match database world, the answer to a query is unanimously defined. Interestingly, there seems to be no single definition regarding how preferences should be interpreted in the context of database queries.

One reason is that existing approaches have thought of preferences as a means of solving seemingly different problems. Driven by the empty-answer problem, several approaches treat preferences as soft constraints that are part of the query [4, 11]. In contrast to the classical hard constraints of the query, soft constraints may be optionally satisfied in the final answer. Dealing with the too-many-answers problem, other approaches view preferences as additional constraints that can be applied in conjunction with the query conditions to prioritize and possibly restrict the query results [14, 29]. However, more importantly, the lack of a single preference-aware query model is due to the fact that preferences can be defined either qualitatively or quantitatively and different approaches for embedding qualitative or quantitative preferences into queries have emerged.

If preferences are expressed in a qualitative way (i.e., using preference relations), then they are embedded into relational query languages through an operator that selects from its argument relation the set of the most preferred tuples according to a given preference relation. This operator has been given different names such as winnow [4], BMO [11] and skyline [2]. The set of most preferred tuples is typically defined as the set of all tuples not dominated by any other tuple. Many variations of this definition have been proposed mainly due to the computational complexity of the optimal answer based on the original definition. For instance, according to the k-dominant skyline definition, a tuple t_i k-dominates another tuple t_j if there are k dimensions, or preferences, in which t_i is better than or equal to t_j and t_i is better in at least one of these k dimensions [3]. Ranking the whole input can be achieved by repetitive applications of this preference operator.

On the other hand, if preferences are expressed quantitatively (i.e., using scores) and there is an aggregating function for combining the partial scoring predicates, then tuples are assigned scores, and the answer of a query with preferences is defined as the ranked set of top-n results [5, 9]. A complementary approach is to make sure that the query results satisfy at least some of the specified preferences [14].

There has also been a significant number of proposals for extending SQL with special preference operators. For example, skylines were first introduced in SQL using the following clause [2]:

SKYLINE OF A_1 [MIN | MAX | DIFF], ..., A_m [MIN | MAX | DIFF]

where a MIN (or MAX) specification following an attribute A_i expresses preference of small (or large) values of A_i whereas DIFF expresses that only tuples with identical values of A_i are comparable.

The top-n preference operator was introduced in SQL by adding a clause of the form:

ORDER BY [user-defined-function] STOP AFTER n

In this case, the tuples in the result are ranked according to a user-defined scoring function and the results with the n highest scores are returned.

As yet another example, Preference SQL supports soft constraints using a special PREFERRING clause [11]:

PREFERRING [conditions] BUT ONLY [conditions]

Preference SQL follows a "best matches only" (BMO) semantics. First, it finds all perfect matches to the conditions of PREFERRING. If this set is empty, then it considers all other best matching values excluding those that do not satisfy the BUT ONLY conditions.

As a result of the proliferation of different interpretations of preferences in the context of database queries, no single preference-aware query model has been widely adopted in the database world and implemented as an alternative or an extension of the exact-match database query model. As a first step towards reaching consensus, we need to understand that we do not try to solve two disconnected problems by integrating preferences into queries: both the empty-answer and the too-many-answers problems are two sides of the same coin caused by the strictness of the exact-match query model. One can go from an empty answer to too many results and vice versa. For example, integrating preferences as additional constraints to restrict a query with too many answers may lead to an empty answer. Bringing preferences into queries is a means to 'relax' the exact-match query model and tackle both these problems in a unified way.

More importantly, a widely accepted preference-aware query model needs to transparently extend the exact-match model with the notion of preferences. One challenge in designing such a model is to come up with new query components that capture the main concepts of preference-aware query answering without adding too much complexity. Ideally, these components should be oblivious to how preferences are represented or implemented and they should naturally blend with the traditional query constructs. In addition, a preference-aware query model should be flexible regarding how preferences are treated within a query. For instance, all preferences could be considered optional or a lower bound on the number of preferences that should be satisfied in the results could be provided by the user.

At a high level, we envision that a preference-aware query would contain the following parts:

- *hard constraints* that are exactly matched
- *soft constraints* (i.e., preferences) that are optionally satisfied. If a query has no soft constraints then we roll back to the exact-match query paradigm, where the answer satisfies exactly the hard conditions.
- a *preference bound* that specifies how preferences should be treated: an *any preference* option would allow any answer that matches the hard constraints (if exist) and any preferences (e.g., as in [11]) whereas an *at least N preferences* option would restrict candidate answers to those that satisfy at least *N* preferences (e.g., as in [14]).
- a *best-answer specification* that describes which of the candidate answers should be output. For instance, a *best-n* answer would comprise the best *n* tuples. If n=1, then only the tuples that are of the highest interest to the user would be returned.

The more the new query components are sealed from the details of their actual implementation, the easier is for users to understand and formulate queries with preferences. For instance, the *best* answer may vary and different algorithms may be required for answer computation depending on whether preferences are quantitatively or qualitatively captured. However, when formulating a preference-aware query, a user just needs to describe how much of the answer she is interested in seeing without caring about such details. This is also true, to some extent, for traditional queries: for example, a user does not care how a join is executed. Finally, a transparent preference-aware query model makes application development easier since the interface to a preference-aware database can be standardized even if the underlying implementation of the engine may vary.

4 Implementations of Preference-Aware Query Answering

In implementing preference-aware query processing, there are three possibilities: (*i*) query translation, (*ii*) special evaluation algorithms for implementing preference operators on-top of the DBMS, and (*iii*) native implementation, i.e., inside the database engine using specific physical operators and algorithms. We call the first two approaches plug-in since they are implemented on top of the database engine.

The straightforward approach to preference-aware query processing is to translate queries with preferences to conventional queries and execute them over the DBMS. Query translation conceptually involves the following steps (e.g., [4, 12, 14]): (*query rewriting*) the preferences are integrated as standard query conditions in the query producing a set of new queries, (*materialization*) the new queries are executed and (*aggregation*) the partial results are combined into a single ranked list. This is for example the approach taken in the implementation of Preference SQL, where preference queries are translated into standard SQL queries [12] as well as for integrating user preferences into queries for personalizing query results (e.g., [14]).

There is a large number of special evaluation algorithms for implementing preference operators (e.g., top-*n* [10] and winnow [2, 4]). For retrieving the top-*n* tuples of a relation, the basic idea has been captured in the algorithm called *FA* [5] and has been improved by several subsequent algorithms (e.g., [9]). These algorithms can be thought of as operating on the aggregation phase of query translation [16]. *FA* performs a sorted access to each partial list of results until there is a set of tuples, such that each of these tuples has been seen in each list. Then, for each tuple seen, *FA* performs random accesses to retrieve its missing scores and compute an aggregate score. Finally, the top-*n* tuples based on their aggregate scores are selected.

The naïve way to compute the winnow of a relation is to apply a basic nested-loop method that compares each tuple in the relation with every other tuple. This method works for every type of preference relation but requires scanning the whole relation for each tuple, which becomes inefficient for large datasets. Several improved algorithms have been proposed over this basic version (e.g., [2]) that employ techniques such as indexing (e.g., [21]) to avoid scanning the whole relation.

These special evaluation algorithms are typically implemented on top of the DBMS on the returned query results as standalone programs. In order to facilitate the integration of any preference operator (e.g., top-*n*, skyline) into the database engine with minimal effort, FlexPref is a system that allows registering preference methods through a set of functions that define rules for determining the most preferred tuples [15]. Once a preference method is injected into the database, it can be used for querying the database.

In the case of implementing preference operators inside the database engine, a set of algebraic rules that characterize the interaction of winnow with the standard relational operators, such as commutativity, are provided in [4] and can be used for query optimization. However, few actual native implementations exist. The rank operator [16] is an extension to relational algebra that enables pipelining and hence optimizing top-*n* queries. A hybrid approach to implementing Preference SQL that provides support for preference-aware query processing within the database engine has been also proposed [7].

Both native and plug-in approaches have pros and cons. Plug-in approaches rely on the existing Boolean query model, which is preference-agnostic. Their main advantage lies in the fact that they require no modification of the database code. They are easier to implement, and they leverage existing, invested, technology. On the downside, treating the DBMS as a black box means that there is no access to internal database operators. Consequently, these methods do not lend themselves to finer-grained dynamic optimization at the operator level based on the features of the query and the preferences. This has an impact on query performance and scalability.

In particular, in query translation, the execution plan for a query with preferences is composed outside the database engine. This external query plan is a high-level execution plan that describes the queries sent to the database, materializations of intermediate results and how partial results are combined in order to generate the answer to a query with preferences. Inevitably, possible query optimizations are more coarse-grained such as reducing the number of queries sent to the DBMS or the data processed outside the DBMS engine. Implementations of preference operators as programs outside the DBMS work as a post-processing step after partial query results have been generated, i.e., primarily improving the aggregation phase.

A fully native implementation of preference-aware query processing requires modifying the database query engine so that queries with preferences can be directly expressed in the query model supported by the database itself and query processing is entirely pushed inside the database engine. Native implementations of operators such as the rank operator [16] represent a significant step towards this direction. However, they mainly aim at

Figure 2: Preference-aware query plans

pushing aggregation inside the database engine while preference evaluation may be still performed externally.

To illustrate the benefits of a fully native implementation, let us assume a preference-aware DBMS. A hypothetical internal query plan (along the lines of the query model outlined in Section 3) is shown in Fig. 2(a). There are two preference-related operators combined with the classical relational operators. The 'preferring' operator applies a preference on a relation. For instance, if we have a quantitative preference that associates comedies with a score 0.8, the 'preferring' operator would assign this score to comedies. The 'best(n)' operator selects the *n* most preferred tuples. Such a query plan can be dynamically optimized taking into account the properties of each operator. For example, we could push the filter down the tree, apply the most selective preference first (that is the one referring to actors) or apply the 'best(n)' operator to each of the input relations to restrict the size of intermediate results. Fig. 2(b) illustrates some possibilities.

Another benefit of a native implementation is that query expressivity is not restricted by having a preference-agnostic database model hidden behind a query translation or special evaluation mechanism. Instead, queries with preferences are precisely defined by the preference-aware query model supported by the database. An important challenge is how expressivity and performance can be balanced. Depending on whether a qualitative or quantitative approach is followed, the new operators may have different properties enabling different optimizations. Making the operators very generic may limit the range of optimizations and may weigh down the benefits of a generic implementation. A careful design of the query optimizer is required.

5 Personalized Queries

In a preference-aware database, queries with preferences could be explicitly issued by users. For example, we can imagine a user interface that allows users to specify hard and soft criteria for movies they would like to see. Querying with preferences is a more relaxed search paradigm but it does not ensure that empty or too many answers will not be generated. To a great extent, it is still up to the user to formulate the right queries. In addition, user preferences may be ephemeral coming along with a particular request or long-term. For example, Bob is not a fan of R. Zelweger. This could be considered a long-term preference. A personalized preference-aware database system would store and maintain user preferences along with data. Furthermore, given a query and an ephemeral or long-term user profile, a personalized DBMS would dynamically decide how to best answer the query in order to satisfy the user information need. For instance, it could dynamically decide that for query Q_1 it does not make sense to show movies with actress R. Zelweger to Bob. Given the proliferation of applications that rely on user preferences to provide a personalized experience, there is an ongoing need for such personalized database systems that provide storage and processing of user preferences.

Dynamically modifying a query taking into account user preferences is termed *query personalization* [14]. Existing works implement personalized queries on top of the exact-match query model rather than a preference-aware one. Their most important limitation is that the effect of query personalization is manually determined

(e.g., by specifying the number of preferences that the answer should satisfy [14]) and they focus on how to retrieve these preferences from the profile and generate the expected personalized results.

However, query personalization needs to be considered from a fresh perspective as a complex dynamic optimization problem. There are several possible queries that can be generated and executed in place of the initial user query by applying query transformations such as: (*i*) some of the hard conditions of the query may be converted to soft conditions; (*ii*) preferences may be added in the query; (*iii*) preferences may replace some of the hard conditions. Which personalization option to follow, what preferences to consider related in the context of a particular query and what makes the best answer to a query for a particular user are shaped by several factors. The DBMS needs to consider the characteristics of the query and the possible effect of the preferences on the query. For example, if a query returns just a few results (like a query about movies with Meg Ryan), then user preferences may be applied only for result ranking. For a too broad query (e.g., a query on comedies), query personalization may use preferences to focus the result set. However, not all preferences may be applicable. For instance, preferences related to dramas may lead to empty answers if combined with a query about comedies. The query context is also important. For example, when a user is browsing movies, query personalization may be less aggressive than when the user explicitly asks for personalized suggestions.

Ideally, a personalized database system should help users find interesting answers that are natural given their queries. Answers that cannot be justified or explained in the context of their initial query or seem random are bound to confuse users. For example, if the user has asked for new comedies she has not watched, then if there are no such comedies, it may make sense to show zero results than suggesting watching a drama. Finally, some control may be given to the user at the application level in order to express the desired extent of personalization.

6 Conclusions

Querying with preferences provides a relaxed query paradigm compared to the exact-match database query model and can help generate answers tailored to each individual. There is substantial work in this area but no single preference-aware query model has been widely adopted in the database world. We believe that the key to making this happen is coming up with a preference-aware query model that is simple and as transparent to preference representation and implementation details as possible. Furthermore, in order to fully unleash the power of a preference-aware query model and exploit query optimization opportunities in the database core, such a model should be natively supported by the database engine. Pushing preferences inside the DBMS will also provide a transparent unified interface to applications and make application development lighter since much of the preference-related logic will be supported by the underlying database system.

Embedding preferences in database queries is a big step forward but alone it will not help users find the ‘right’ answers to their queries. A personalized database system would actually help by cleverly modifying queries considering user preferences, whenever available, to generate customized answers. However, from a traditional DBMS to a personalized preference-aware DBMS, there is a long way to go, and it goes beyond disguising the elephant to make it look like a chameleon.

References

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

[2] S. Börzsönyi, D. Kossmann, and K. Stocker. The skyline operator. In *ICDE*, pages 421–430, 2001.

[3] C. Y. Chan, H. V. Jagadish, K.-L. Tan, A. K. H. Tung, and Z. Zhang. Finding k-dominant skylines in high dimensional space. In *SIGMOD*, pages 503–514, 2006.

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

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

[6] P. C. Fishburn. Preference structures and their numerical representations. *Theoretical Computer Science*, 217(2):359–383, 1999.

[7] B. Hafenrichter and W. Kießling. Optimization of relational preference queries. In *ADC*, pages 175–184, 2005.

[8] S. O. Hansson. Preference logic. *Handbook of Philosophical Logic (D. Gabbay, Ed.)*, 8, 2001.

[9] I. F. Ilyas, W. G. Aref, and A. K. Elmagarmid. Supporting top-k join queries in relational databases. In *VLDB*, pages 754–765, 2003.

[10] I. F. Ilyas, G. Beskales, and M. A. Soliman. A survey of top-k query processing techniques in relational database systems. *ACM Comput. Surv.*, 40(4), 2008.

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

[12] W. Kießling and G. Köstler. Preference SQL - design, implementation, experiences. In *VLDB*, pages 990–1001, 2002.

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

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

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

[16] C. Li, K. C.-C. Chang, I. F. Ilyas, and S. Song. Ranksql: Query algebra and optimization for relational top-k queries. In *SIGMOD*, pages 131–142, 2005.

[17] J. Pitkow, H. Schtze, T. Cass, R. Cooley, D. Turnbull, A. Edmonds, E. Adar, and T. Breuel. Personalized search. *Comm. of the ACM*, 45(9), 2002.

[18] K. R. Scherer. What are emotions? and how can they be measured? *Social Science Information*, 44:695–729, 2005.

[19] K. Stefanidis, G. Koutrika, and E. Pitoura. A survey on representation, composition and application of preferences in database systems. *ACM Trans. Database Syst.*, 36(4), 2011.

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

[21] K.-L. Tan, P.-K. Eng, and B. C. Ooi. Efficient progressive skyline computation. In *VLDB*, pages 301–310, 2001.

[22] M. P. Wellman and J. Doyle. Preferential semantics for goals. In *Proc. of AAAI*, pages 698–703, 1991.

[23] X. Zhang and J. Chomicki. Preference queries over sets. In *ICDE*, 2011.

Preferences and Attitudes for Personalized Information Provision

Yannis Ioannidis, Maria Vayanou, Katerina Iatropoulou, Manos Karvounis, Vivi Katifori,
Marialena Kyriakidi, Alexandros Mouzakidis, Natalia Manola, Lefteris Stamatogiannakis,
Mei Li Triantafyllidi
MaD*g*IK Lab, Dept. of Informatics & Telecom, University of Athens, Hellas (Greece)
{yannis, vayanou, kiatrop, manosk, vivi, marilou, alexm, natalia, estama, meili}@di.uoa.gr
http://www.madgik.di.uoa.gr/

Abstract

PAROS is a generic system under design whose goal is to offer personalization, recommendation, and other adaptation services to information providing systems. In its heart lies a rich user model able to capture several diverse aspects of users behavior and users interests, preferences, and attitudes. The user model is instantiated with profiles of individual or group users, which are obtained by analyzing and then appropriately interpreting potentially arbitrary pieces of user-relevant information coming from diverse sources. These profiles are maintained by the system, updated incrementally as additional data on users becomes available, and taken advantage of by a variety of applications to adapt the functionality of information systems to the users characteristics.

1 Introduction

The quality of human interactions is often intimately affected by the level of understanding that the persons involved have for each other. To a large extent, this holds for interactions between persons and information-providing systems as well. This paper outlines the philosophy and general approach of PAROS, a system under design and initial development at the Univ. of Athens, whose goal is to obtain such an understanding in the form of profiles that it maintains for its users so that it may offer information and other services to them in a personalized and adaptive fashion. The system aims to provide rich relevant functionality at a generic level, on top of which, particular applications may be built. The key contribution of our work around PAROS is fourfold:

(a) User-model (profile) independence: Currently, personalization and other user-based adaptations are mostly offered in a tightly coupled fashion with the main information system or application concerned. They are based on custom user models and profiles that are intimately dependent on the specific application domain and profiling is based on user interactions with that same system itself. In reality, however, users typically interact with a variety of systems and applications, and their behavior in each environment is useful not just for that one but for all. PAROS moves away from this restrictive practice and separates the user model and its associated profiles from the profiling and the preceding user-data analysis techniques, on the one hand, and the application adaptation strategies, on the other. Hence, ones profile may be synthesized from information coming from several applications and be used to adapt the behavior of several other applications, achieving the desired independence to a large degree.

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

(b) User-model expressive power: As a consequence of their application dependence, current user models often deal with a single type of user preferences or attitudes (e.g., just behavioral tendencies, such as like to watch or like to read) for just a single class of entities (e.g., just movies or just books). PAROS supports a labeled-graph-based user model that permits, in principle, any kind of attitudinal tendency towards any entity managed by the applications interacting with the system. Furthermore, the users, the world of relevant entities and relationships, and most importantly, the various kinds of attitude and their intensity are modeled in a uniform fashion, allowing for the effective expression of interactions and influences between them.

(c) Profile-derivation foundations: Derivation (prediction) of unknown user attitudes is a major part of most related efforts contributions and is usually approached in seemingly ad hoc ways, unrelated to each other. Based on the graph form of its user model and the resulting profiles, PAROS formulates most types of profile derivation as path computations on these graphs. Having this common foundation, it is able to capture a wide variety of approaches by simply choosing different functions for the two function-valued parameters that lie at the heart of the definition of path computations. Thus, the effectiveness of each approach depends on the cognitive and psychological plausibility of these two functions, while its efficiency depends on the algebraic properties of these functions and the performance of the suite of available path computation algorithms.

(d) Application environment novelty: The flexibility and richness afforded by the design of PAROS have inspired relevant work in several novel and challenging environments, whether with respect to profiling or to application adaptation or both. Traditional web logs recording user actions, large discussion threads, multimedia file streams, and user traces from a variety of sensors (e.g., geo locations from smart phones) are some of the sources we are dealing with for user profiling in different contexts. Recommendations in digital libraries, personalized storytelling during museum visits, and opinion summarization on sensitive divisive issues are some of the application environments we are working on.

Each of the above aspects of PAROS presents significant new research (and engineering) challenges, which are part of our current and future work. In addition, the old and ever-present issue of heterogeneous data integration or harmonization arises in every PAROS component, due to the syntactic and semantic diversity of data that are part of user profiles, serve as input to the profiling processes, or are otherwise managed by the applications, but that is not a focus in the context of this work. The remaining sections of the paper are in more or less one-to-one correspondence with the four contribution areas above, identifying the relevant challenges and highlighting the corresponding directions of our work.

2 System Architecture And User Model Independence

Figure 1 presents the key system components and their interactions. ***User Model and Profile Management*** serves as the foundation of PAROS on top of which all other parts are built. It provides the basic functionality required by the PAROS user model, which captures several levels of user behavior, user preferences, and attitudes towards the relevant world entities and concepts. In the database tradition, such functionality may be partitioned into *profile definition* and *profile instantiation*. The former is reminiscent of database schema definition and consists of defining types of attitude to be captured in a particular environment, functions that are available for use in profile derivation, and classes of users, relevant world entities, and their relationships (if any). The latter is reminiscent of database manipulation and consists of storing and maintaining the actual profiles, expressing particular users having particular preferences and attitudes towards particular world entities. ***Pattern Extraction*** includes techniques that analyze user-behavior data and compute statistical measures, correlations, and other usage patterns. Such data may be collected explicitly, by having users provide related input directly, or implicitly, by having the system monitor and document their behavior in different contexts. ***Profiling*** encompasses mechanisms for user-profile creation based on the patterns identified. The latter are appropriately interpreted within the framework of the user model to obtain profile elements, which may then be recursively processed further to expand profiles with additional elements. Both profile interpretation and expansion is done,

ideally, according to cognitive and psychological theories. ***Adaptation*** represents the actual use of profiles to personalize the content provided by information systems and other applications and, in general, to adapt their behavior to the users preferences and interests. In principle, a separate component is required for each external system customized, although certain commonalities may be taken advantage of.

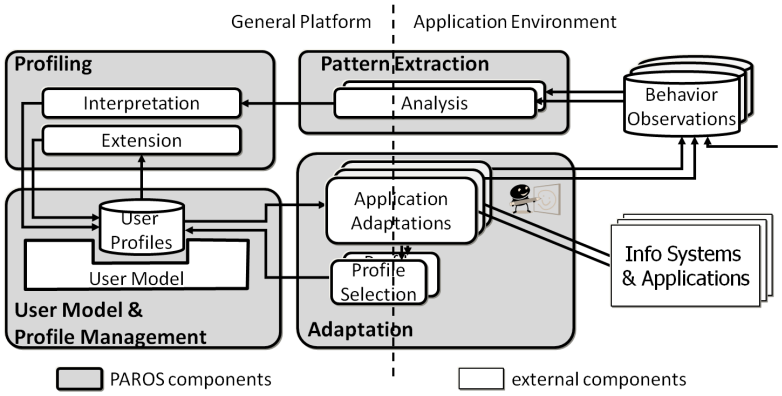

Figure 1: PAROS Architecture

Separation of the four main conceptual components of PAROS above is the critical element that leads to the desirable user model (and profile) independence. User profile elements may come from multiple sources (from applications customized by PAROS or possibly from completely external ones), through different interaction modes (direct user input or indirect user behavior), and after different statistical analysis and other processing techniques, but they are all captured in a unified profile that captures all aspects of users attitudes. This one profile can then be used to adapt any relevant application, independent of its role in shaping up that profile.

3 User Model

The basis of any personalization system is its user model, which must be able to represent in a sufficient level of detail the objects, concepts, and relationships on which any application would offer personalized services. The PAROS user model attempts to represent the entire universe of users and their attitudes in a uniform way, including i) individual users, groups of users, communities, and their relations, ii) entities that are the objects of users attitudes and their relationships to other entities, categories, or concepts, and iii) user preferences and attitudes towards these entities. Below we present a somewhat simplified version of that model removing some complications that are unnecessary in the rest of the exposition but could make it harder to follow. The user model is a graph GUM(UOF,DAPA), where UOF is the set of nodes in the graph and DAPA is the set of edges. Nodes in set UOF primarily represent ***Users*** of the information captured by the system, ***Objects*** of the world, ***Functions*** between the above (also captured in edge set DAPA), and classes of the above (if any). Nodes may also represent particular user groups or communities, object sets, function families, or their combination, if such aggregations play particular roles in the environment concerned.

Edges in set DAPA represent two main kinds of relationships among the graph nodes. The first kind is ***Data*** relationships, as those typically found in databases or other data collections, capturing object structure, association, membership, and inheritance. The second and most interesting kind is ***Attitudinal*** and ***Preference*** relationships, whose corresponding edges emanate from user nodes and are incident to any kind of node. Examples include attitudes towards users (e.g., *trusts* a friend, *is inspired by* a celebrity, *follows* an authority), world objects (e.g., *likes* a concrete object, *supports* a cause, *believes* an ideal), or functions (e.g., *values* a function, in the sense that a preference/attitude towards one of its ends affects the preference/attitude towards the other).

Figure 2 gives a simple example of a profile with these kind of relationships. In principle, there is a third kind of relationships in DAPA, capturing ***Actions*** of users towards various kinds of objects, whether in the real world or in their representation in the system. Examples include actions towards users (e.g., *tag* a friend, *respond to* a forum member), actual world objects (e.g., *buy* a book, *watch* a movie), or virtual objects (e.g., *query* a term, *visit* a page, *send* a comment). Note that functions have a dual representation in GUM, being both edges and nodes, a flexibility that facilitates the expression of certain preferences and attitudes.

Each edge in DAPA potentially has a label vector associated with it, whose structure depends on the edge type. An example of a potential singleton label vector is *trusts*[*ti*](u,v), where for users u and v, trust intensity *ti* $\in$ [-1,1] captures the level of trust that u has for v, where 1 indicates complete trust, -1 indicates complete distrust, and 0 means indifference. Similarly for other attitudinal edges, such as *likes*[*li*](u,o), capturing the level of likeness that user u has for object o, and *values*[*vi*](u,f), capturing the level of importance that user u places in function f for calculating *li* of *likes*(u,o) based on the *li* of *likes*(u,o), for any o, o such that f(o)=o. These label vectors could be enhanced with at least three other elements, [*co*, *so*, *tmp*]. Element confidence *co* $\in$ [0,1] captures the confidence level for the value of the appropriate intensity (*ti*, *li*, or vi*)*, element source *so* indicates where intensity and confidence values have been extracted/derived from, and element timestamp *tmp* indicates the time period during which the other elements of the vector are valid. To a large extent, the structure, semantics, and detailed mechanisms of the label vectors determine the complexity of the overall user model. Simpler or more complex label vectors result in correspondingly less or more refined representations of users preferences and attitudes.

The above model is rather general and can capture most models appearing in the literature (even if using a different notation and representation languages). This generality allows sound experimentation with and comparison of different models and possible future enrichments while remaining in the same framework. In the rest of the paper, we use examples with a relatively basic user model: few action and attitudinal edges (*queries and trusts, likes, values*), and only singleton label vectors with the corresponding intensity levels. Figure 2 gives a simple example of a profile with these kind of relationships.

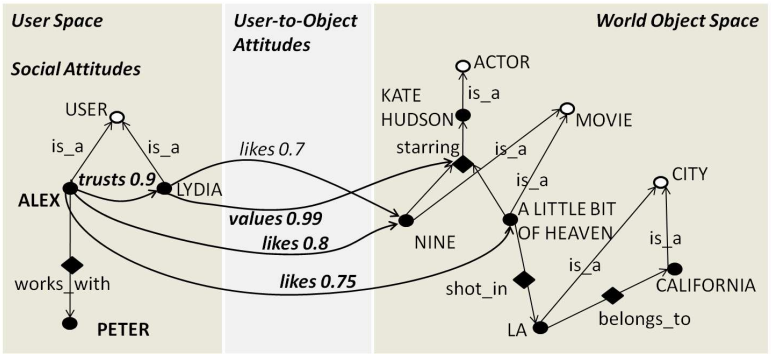

Figure 2: Profile example in the GUM graph user model

4 Profiling Foundations

Profile derivation can be distinguished into first-level derivation (**profile interpretation**, from action edges to attitudinal edges) and second-level derivation (**profile expansion**, from existing attitudinal edges to other unknown attitudinal edges). As a simple example of interpretation, consider two nodes, user Alex and actor Kate Hudson, and a behavioral edge between them, ***queries***[*37*](Alex,Kate Hudson), indicating that Alex has queried about Kate Hudson 37 times. The process of interpretation (of attitudes from actions) could derive an

attitudinal edge between the same nodes, predicting the intensity of Alexs attraction to the said actress based on his behavior related to her: ***likes***[*0.85*](Alex,Kate Hudson). Why 37 queries imply an 0.85 level of attraction (both numbers chosen arbitrarily here, for the sake of the example) is a matter of cognitive and psychological theory, which should be codified in the derivation process in some fashion.

Likewise, as an example of expansion, consider the following sequence of attitudinal and data edges and their end nodes: ***trusts***[*0.9*](Alex,Lydia), ***likes***[*0.7*](Lydia,Nine), ***values***[*0.99*](Lydia,starring), starring(Nine,Kate Hudson). Informally and while ignoring some sticky points, the last two edges, ***values***[*0.99*](Lydia,starring), starring(Nine,Kate Hudson), could be merged into one, ***values***[*0.99*](Nine,Kate Hudson). Together with the first two edges above (***trusts*** and ***likes***), this forms a path through nodes Alex, Lydia, Nine, and Kate Hudson. The process of expansion (of attitudes) could essentially derive a (probably missing) attitudinal edge between the two end nodes, Alex and Kate Hudson, predicting the intensity of the formers attraction to the latter based on indirect information: ***likes***[*0.62*](Alex,Kate Hudson). Again, the methods that generate 0.62 as the attraction intensity should be scientifically justified and be realized in appropriate system components.

Independent of its level, derivation in PAROS is abstracted as an optimal path computation problem, similar to the shortest path, most reliable path, or bill of materials problems. All these problems are defined on labeled directed graphs and essentially define a property for an (existing or non-existing) edge (a,b) based on corresponding properties of all the paths between nodes a and b in the graph. Most path computation studies use a path algebra formalism [1]. Informally, there is a label associated with each edge and each path in the graph. A path label is computed as a function, called *CON* (for concatenate), of the sequence of labels of the edges in the path. In addition, a path set P is also associated with a label, which is computed as a function, called *AGG* (for aggregate), of the labels of the paths in P.

Profile derivation does not directly map on to conventional path computations, having several additional complications or differences that raise some new theoretical and computational challenges. For example, edges typically have rich label vectors instead of just individual scalar values (e.g., labels could be of the form [intensity, confidence, timestamp,]), edges are of different types and affect each other in diverse ways (e.g., different combinations of attitudinal-edge types may be used to derive further edges of these or other types), or hypergraphs may actually be necessary to model the problem (e.g., in principle, the edge ***values***[0.99](Nine,Kate Hudson) in the example above is a hyperedge ***values***[0.99](Lydia,Nine, Kate Hudson), as its source depends both on the user (Lydia) and the movie (Nine). Still, modeling derivation as path computation and using that as the foundation of this work is very valuable and serves as a starting point in our current and future efforts.

5 Adaptation Applications

In this section, we report on several interesting application environments and information systems we have worked on to personalize their behavior and make their services adaptive. In each case, we point out the most characteristic aspects of our work, whether in the analysis or the adaptation components of PAROS. The great variety of these environments offers a good indication of the applicability of our overall approach.

Query Recommendations in Digital Libraries: In the context of the portal of European National Libraries (TEL [2]) , we have implemented a suite of mining techniques to analyze the usage logs maintained by the portal, including techniques for data collection, log cleaning, log transformation, and pattern extraction. All these form appropriate workflows of data processing operators that are built on top of and orchestrated by **madIS** [3], an open-source extensible data transformation and analysis system that we have developed for this purpose, but is proving a reliable platform in other ongoing efforts as well. In addition to the intrinsic value of the statistical patterns being extracted by the analysis workflows on how the portal is used, they have also formed the basis for identifying personal, national, and global profiles for the users, based on which, the portals services have been extended with recommendations on term queries, library collections, and even similar users[4].

Opinion Mapping in Web Discussion Forums: Our work in the area has focused on analyzing online

discussions, summarizing the key opposing viewpoints, and profiling the discussants in terms of the side they support. We have used the posts of a discussion thread to construct a Reply Graph, where nodes denote posts and an edge from post A to post B indicates that A quotes B and replies to it. Careful analysis of this graph leads to *discussion topic extraction* and *like-minded discussant identification*. The topic extraction analysis algorithm is based on identifying the appropriate keywords within the scope of each post, which consists of the posts that are closely related to it, whether topologically (proximity in the Reply Graph) and chronologically (proximity of the posts timestamp). The end result is the set of topics of every post, even of small posts with no characteristic keywords in their bodies, and the overall set of topics of each discussion and sub-discussion. The group identification analysis algorithm is based on the observation that most discussion exchanges are expressing disagreement [5] [6] [7]. The end result is the set of discussant groups, where each group has a particular opinion or attitude for the discussion topics and members of each group largely disagree with members of other groups.

Other applications: In addition to ongoing work in the above two environments, we are also involved in some other efforts that take advantage of the PAROS approach but also enrich its range of applications. One such effort investigates the basic characteristics of trust between people and also its interpretation and expansion based on social interactions in social networking sites. Another one is building on madIS to monitor scientists publications life-cycle, form the profiles of ad hoc research communities, and provide personalized search and alerting services to them. Finally, an interesting effort focuses on personalized and adaptive storytelling during museum visits, where visitors profiles will be derived based on the route they follow while in the museum, details of their interaction with relevant applications on their mobile phones, and stereotypical information about them, so that the story narrated guiding them through the exhibits is dynamically adapted to those profiles.

6 Conclusions and Future Work

Based on the concept of user model independence, its general graph-based user model, and its use of path computations as the starting point for profile derivation, PAROS aims to become a comprehensive platform for user modeling, user profiling, and profile-based adaptation of information provision and other services. In addition to investigating the limits of the overall approach, the system will serve as a platform for studying the cognitive and psychological validity of profile derivation techniques, the scalability and efficiency of behavior analysis and pattern discovery algorithms, the effectiveness of adaptation strategies, and other relevant research issues. Design and implementation of PAROS are currently under way and initial experimentation with available parts is quite promising.

References

[1] B. Carre, “Graphs and Networks”, Oxford: Clarendon Press, 1979.

[2] http://www.theeuropeanlibrary.org

[3] http://code.google.com/p/madis/

[4] M. Kyriakidi, L. Stamatogiannakis, M. L. Triantafyllidi, M. Vayanou, Y. Ioannidis, “A Prototype Personalization System for the European Library Portal”, in *European Conf. on Digital Libraries*, 2010.

[5] M. Karvounis, T. Georgiou, Y. Ioannidis, “Utilizing the Quoting System of Online Web Forums to Estimate User Agreement”, *SIGMOD Undergraduate Research Poster Competition, ACM SIGMOD/PODS*, 2010.

[6] T. Georgiou, M. Karvounis, Y. Ioannidis, “Extracting Topics of Debate between Users on Web Discussion Boards”, *SIGMOD Undergraduate Research Poster Competition, ACM SIGMOD/PODS*, 2010.

[7] R. Agrawal, S. Rajagopalan, R. Srikant, Y. Xu, “Mining newsgroups using networks arising from social behavior”, in *Proceedings of the 12th international conference on World Wide Web*, 2003.

An Overview of the CareDB Context and Preference-Aware Database System

Justin J. Levandoski Mohamed E. Khalefa Mohamed F. Mokbel

Department of Computer Science and Engineering, University of Minnesota, Minneapolis, MN
{justin,khalefa,mokbel}@cs.umn.edu

Abstract

This article provides an overview of the CareDB *context and preference-aware database system.* CareDB *provides efficient and scalable* personalized *query answers to users based on their preferences and current surrounding context.* CareDB *moves beyond the* rigid *query processing semantics of traditional relational database systems, which employ a boolean "all or nothing" query model, and addresses support for "preference-aware" query processing methods. Specifically,* CareDB *supports a plethora of multi-objective preference methods capable of finding the "best alternatives" according to users' given preference objectives. This article describes the architecture of the* CareDB *system and describes the details for three of its novel query processing characteristics: (1) a generic and extensible preference-aware query processing engine, (2) a framework to gracefully handle contextual attributes that are expensive to retrieve, and (3) a framework to efficiently process queries over uncertain contextual data.*

1 Introduction

Currently, database systems are extremely rigid: a user submits a query with a set of constraints to the database, and the system returns a set of answers that are exact matches for the constraints. In the worst case, the database may return no answers if the given constraints are too restrictive. For many application scenarios (e.g., location-based services, point-of-interest finders), users want from the database only a few "best" answers according to their personal preferences and context (e.g., location, weather). For instance, when searching for a restaurant, a user may want the database to return only five restaurants that present the best trade-off between minimizing the price and travel time to the restaurant, while maximizing the restaurant rating.

In the database literature, a number of multi-objective preference methods have been proposed that are capable of evaluating a set of user preference constraints. Examples of these methods include top-k [5], skylines [2], hybrid multi-object methods [1], *k*-dominance [3], *k*-frequency [4], and top-*k* dominance [13]. In general, the point of proposing new preference methods is to challenge the notion of "best" answers. Given the large number of preference methods already proposed, and likely to be created in the future, a fundamental research issue has become how to embed the semantics of each preference method within a database management system that may execute arbitrary queries composed of relational operators (e.g., selection, joins).

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

Another important consideration is how to integrate dynamic *contextual data* into preference query processing. Contextual data refers to any interesting data about the user or her environment that can help refine a preference answer. Currently, such data is readily available through third-party web services. For instance, when searching for restaurants, a user may want to take into account travel time (e.g., using the Bing Maps web service) or restaurant reviews (e.g., using the Yelp web service). While useful, this contextual data poses two main problems when integrated with preference query processing: (1) the contextual attributes are *expensive* to derive relative to static data stored locally in a database (e.g., retrieved from a third party over a network) and (2) contextual data may contain uncertainty (e.g., restaurant prices reported as a range).

Toward the goal of embedding support for context and preference within a DBMS, this article describes *CareDB*: a context and preference-aware database system. *CareDB* addressees two fundamental *core* systems challenges: (1) support for various multi-objective preference evaluation methods (e.g., skyline, top-*k*, *k*-dominance) within the query processor and (2) integration of surrounding contextual data (e.g., current traffic, weather) within the query processing, calling for support for gracefully handling expensive attributes and uncertain data. *CareDB* is a complete database system, meaning all ideas discussed in this article have been realized and experimentally evaluated in the PostgreSQL [12] open-source database system.

In the rest of this article, we describe how *CareDB* addresses the core systems research challenges behind embedding context and preference within the database query processor. Section 2 provides an overview of the *CareDB* architecture. Section 3 describes the novel technical features of *CareDB*. Finally, Section 4 describes a *CareDB* prototype.

2 CareDB Overview

This section provides an overview of the *CareDB* context and preference-aware database system, depicted in Figure 1.

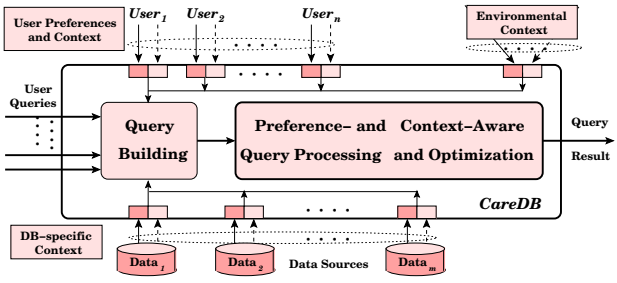

Figure 1: *CareDB* Architecture

Input. Besides queries expressed in SQL, *CareDB* takes *preference* and *contextual* data as input. Preferences are specified by a user and stored in a profile. In *CareDB*, a single user preference maps to single data attribute. The structure of a preference is **`(Attribute, Preference, [Value])`**, where *attribute* is a single data attribute, *preference* describes a user "wish" for that attribute, and *value* (numeric, categorical, or boolean) is optional determined by the type of preference. Preferences for a user are stored in their *preference profile*. The available preferences are based on the theoretical foundation of *PreferenceSQL* [8]. Preferences can be *hard* (e.g., equals) or *soft* (e.g., highest, lowest), and specified over numeric or categorical attributes. Furthermore, the user may specify a ranking function (either user-defined or built-in) over multiple attributes in order to perform top-k preference evaluation.

In addition, *CareDB* has three input context types, each context can be either *static* (rarely changed) or *dynamic* (frequently changing). (1) *User context*. User context is any extra information about a user. Examples of static user context data include income, profession, and age while dynamic attributes include current user location or status (e.g., "at home", "in meeting"). (2) *Database context*. Database context refers to data sources (e.g., restaurant, hotel, and taxi databases) that are registered with *CareDB*, representing data in the domain a user wishes to query. As an example, for a restaurant database, static context data includes price, rating, and operating hours while dynamic context includes current waiting time. (3) *Environment context*. Environment context is any information about the user's surrounding environment, assumed to be stored at a third party and

consulted by the query processor. A dynamic environmental context includes road traffic, while a relatively static context includes weather information.

Query Building Module and Preference Queries. The purpose of the query building module (rounded square in Figure 1) is to *personalize* queries for each user such that the *best* answers are returned. The user submits simple SQL queries without constraints (e.g., "Find me a restaurant"). The query building module creates *preference queries* by augmenting the submitted query with the preference constraints stored in the user's preference profile. *Preference queries* contain traditional relational constraints (e.g., selection conditions), as well as new preference constraints added to a preferring clause. In general, hard preference constraints are added to the where clause while soft preference constraints are specified in the preferring clause. Meanwhile, a using clause specifies what preference method will be used to evaluate the preference constraints to produce an answer. An example preference query for restaurants is given in Figure 2 that employs the skyline method in the using clause to produce a preference answer.

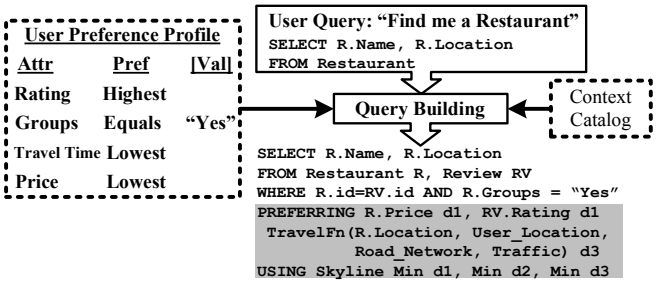

Figure 2: Preference queries in *CareDB*

Query Processor. The preference and context-aware query processing and optimization engine is responsible for executing preference queries in *CareDB*. The main novelty of *CareDB* lies in this query processing engine. The main responsibilities of this module are as follows. (1) Embed various types of preference-aware query processing within a relational database engine. Specifically, we aim to support various types of multi-objective preference methods, e.g., skylines [2], *k*-dominance [3], top-*k* dominance [13]. Each method accepts preference objectives like those specified in the preferring clause of the query in Figure 2. However, given the same constraints, each method may produce a different preference answer for a given data set. (2) Support the integration of context-aware query processing. Contextual data (e.g., traffic and weather information) is assumed to be retrieved from a third-party source and expensive to derive relative to data stored locally in the database. (3) Support preference and context-aware query evaluation that involves *uncertain* data.

3 CareDB Technical Features

A distinguishing feature of *CareDB* is its integration of preference and context concepts *inside* the database query processor. In this section, we describe the details of three novel systems features of *CareDB*: (1) *FlexPref*: the generic and extensible preference query processing engine of *CareDB* that includes an efficient preference-aware join operator, (2) a preference query processing framework for efficiently handling computationally-bound contextual data, and (3) a framework for efficiently answering preference queries for uncertain data.

3.1 FlexPref: An Extensible Preference Query Processing Framework

CareDB queries can be evaluated using any number of preference methods (e.g., skyline, top-k, k-dominance) based on the constraints given in the preferring clause. Thus, the query processor must be aware of how to evaluate any of these methods. One approach is to create a user-defined-function that evaluates preference *on-top* of a query plan. A second approach is to create a *custom* implementation for each preference method that can be integrated with query operators. *CareDB* takes a third approach by implementing *FlexPref* [9], a *general* and *extensible* framework for implementing preference evaluation methods inside the query processor. Figure 3 relates the main idea of *FlexPref*. The framework is built into the PostgreSQL query processor, and

only FlexPref touches the query processor. Each new preference method added to the system is "plugged into" *FlexPref* by registering only *three* functions: (1) PairwiseCompare: given two data objects P and Q, update the score of P and return whether P or Q can never be a preferred object based on a pairwise comparison of objects. (2) IsPreferredObject: given a data object P and a set of preferred objects S, return true if P is a preferred object and can be added to S, false otherwise. (3) AddPreferredToSet: given a data object P and a preference set S, add P to S and remove or rearrange objects from S, if necessary.

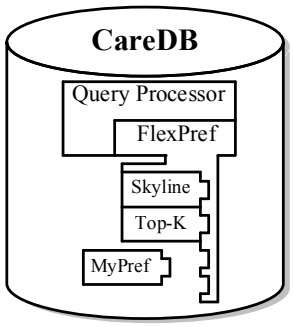

Figure 3: *FlexPref*

The main idea behind FlexPref is *separation of duties*. (1) The *registered functions*, specific to each preference method, define the semantics of the preference criteria. These functions define *how* one object is qualitatively better than the other. These functions are *not* aware of the internals of the query processor. (2) The *generalized* framework is responsible for efficient preference query processing by injecting preference evaluation as close to the native data operators as possible (i.e., selections, joins). The generalized framework uses the registered functions to evaluate the semantics of the specific preference method. With *FlexPref*, a preference evaluation method can "live" inside the query processor with minimal implementation effort compared to a *custom* approach. *FlexPref* consists of a set of generic, extensible relational operators. The basic idea is that each operator is written in terms of the extensible functions. The operator implements the common query processing functionality common across all supported multi-objective preference methods. During query runtime, the appropriate plugin functions for a specific preference method are used by the operators to evaluate the semantics of that method. Currently, *FlexPref* consists of the following three operators.

Selection. The selection operator produces the preference answer from data stored in a single table. The basic idea of the selection operator is to implement a block-nested loop algorithm to execute single-table preference evaluation (assuming data is not indexed). The operator compares tuples pairwise, using the plugin functions to evaluate the specific preference semantics, and incrementally builds a preference answer set.

```
SELECT  R.id, H.id
FROM Restaurants R, Hotels H
WHERE R.city=H.city, R.groups="yes"
PREFERRING R.price rp, H.rating hr,
           R.rating rr
USING KFrequency min rp, max hr,
      max rr with k=2;
```

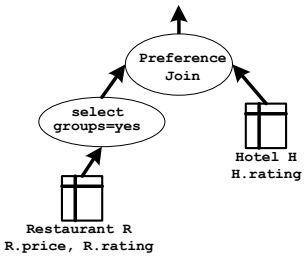

Figure 4: Preference join

Join. *FlexPref* integrates preference evaluation with the join operation, resulting in an operator named PrefJoin [7, 9]. For example, the preference query depicted in Figure 4 involves a join between the Restaurant and Review relations, where preference evaluation is performed using attributes from both relations. A naive method to implement a preference-aware join would be join all necessary data and perform preference evaluation afterward. PrefJoin improves upon this naive strategy by using the extensible plugin functions to prune tuples *before* the join that have no chance of contributing to the final preference answer. The PrefJoin algorithm consists of four phases, namely, *local pruning*, *metadata preparation*, *join*, and *refinement*. The *local pruning* phase filters out, from each input relation those tuples that are guaranteed not to be in the final preference set. The *metadata preparation* phase associates metadata with each remaining tuple used to optimize the join. The *join* phase uses the metadata to decide on which tuples should be joined together. Finally, the *refinement* phase finds the final preference set from the output of the join phase. These steps enhance join performance as joins are usually non-reductive, thus pruning tuples early reduces the output of the join, which in turn means less data must be processed in subsequent operations after the join.

3.2 Query Processing with Expensive Contextual Data

In *CareDB*, it is assumed that some attribute values will be expensive to derive, as the derived value may require extensive computations (e.g., road network travel time), or must be retrieved from a third party (e.g., remote web

service). Figure 5 gives an example query (and plan) to find a preferred restaurant using the top-*k* domination method [13], where attributes *price* and *rating* are stored in a local relation, while the *travel time* attribute is requested from the Microsoft MapPoint [11] web service based on the restaurant and user locations. Under these circumstances, computational overhead is dominated by deriving the expensive *travel time* attribute, thus the preference query processing operator should avoid computing these expensive attributes whenever possible.

```
SELECT R.id, R.Location
FROM Restaurants R,
     MapPoint M
PREFERRING R.price p, R.rating r,
           M.travelTime(R.Location,
                        User.Location) t
USING TopKDomination min p, min t,
      max r with k=10;
```

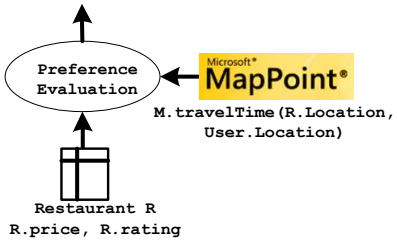

Figure 5: Expensive attribute query

The *CareDB* query processor is designed to take these challenges into account. *CareDB* employs a preference evaluation operator that computes the preference answer by retrieving as few expensive data attributes as possible [10]. The main idea is to first perform preference evaluation over *local* data attributes, forming a local answer set *LA* using "plug-in" functions to determine the semantics of the specific preference method used to execute the query (e.g., top-*k* domination). The operator then selectively requests expensive attributes for objects in *LA* guaranteed to be preference answers, and *prunes* objects that are guaranteed not to be preference answers. *CareDB* then attempts to make the minimum number of expensive attribute requests necessary to completely and correctly answer the preference query.

3.3 Handling Data Uncertainty in *CareDB*

Given the growing number of applications that generate uncertain data (e.g., e.g., sensors, human entry errors), it is likely that some data registered with *CareDB* will contain uncertainty. Thus, *CareDB* employs a query processing framework, named *UPref* [6], capable of answering preference queries over data containing a mix of certain and uncertain attributes. *UPref* assumes uncertain attributes are represented as a continuous range of values, common in many real-life applications (e.g., biological data, spatial databases, sensor monitoring, and location-based services). *UPref* associates a probability *P* with each object *O*, that represents the chance that *P* is an answer to the preference query.

To process queries, *UPref* assumes two system parameters: (1) A *tolerance* value Δ that specifies the maximum error allowed in calculating probability *P*, and (2) a *Threshold* value *H*, that each object probability must exceed in order to be a preference answer. *UPref* employs a two-phase filter-refine approach to processing preference queries over uncertain data. Phase I calculates an estimated upper-bound preference probability for each object, and *filters* objects on-the-fly that have an upper-bound probability that falls below the threshold *H*. Phase II computes a final preference probability for each candidate answer within a user-given tolerance Δ. Phase II employs a novel, efficient probability calculation method that performs only as much computation *as is needed* to guarantee the final preference probability for an object falls within Δ. Much like *FlexPref* and the expensive attribute framework, *UPref* is designed to be generic and extensible, capable of supporting many well-known preference methods within a *single* framework.

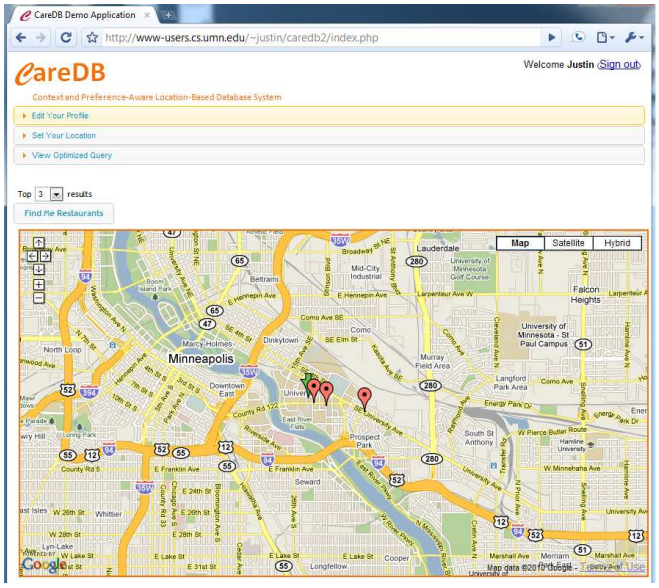

Figure 6: *CareDB* Demo Application

4 CareDB Prototype

To demonstrate the usefulness and functionality of *CareDB*, we implemented a location-based restaurant and hotel finding application using the Google Maps API, depicted in Figure 6, which interacts with the *CareDB* server implemented within PostgreSQL [12]. In the application, users can set their *CareDB* preference profile explicitly using a profile editor window. The editor allows the user to specify their preference objectives, *as well as* the preference method used to evaluate these objectives. Since the *CareDB* query processing framework (i.e., *FlexPref*) is generic and extensible, we provide a number of different preference methods to the user (e.g., skyline [2], top-*k* [5], top-*k* domination [13]). To process queries, the application forwards a simple query to *CareDB* (e.g., "find me a restaurant") where it is injected with preference and context constraints based on the users's preference profile. *CareDB* returns (1) the personalized preference SQL query that was run on the database, which can be displayed the application using a drop-down menu, and (2) the personalized query answers that are displayed on an embedded Google Maps interface.

References

[1] W.-T. Balke and U. Güntzer. Multi-objective Query Processing for Database Systems. In *Proceedings of the International Conference on Very Large Data Bases, VLDB*, 2004.

[2] S. Börzsönyi, D. Kossmann, and K. Stocker. The Skyline Operator. In *Proceedings of the International Conference on Data Engineering, ICDE*, 2001.

[3] C.-Y. Chan, H. Jagadish, K.-L. Tan, A. K. Tung, and Z. Zhang. Finding k-Dominant Skylines in High Dimensional Space. In *Proceedings of the ACM International Conference on Management of Data, SIGMOD*, 2006.

[4] C.-Y. Chan, H. Jagadish, K.-L. Tan, A. K. Tung, and Z. Zhang. On High Dimensional Skylines. In *Proceedings of the International Conference on Extending Database Technology, EDBT*, 2006.

[5] S. Chaudhuri and L. Gravano. Evaluating Top-K Selection Queries. In *Proceedings of the International Conference on Very Large Data Bases, VLDB*, 1999.

[6] M. E. Khalefa, M. F. Mokbel, and J. J. Levandoski. Skyline Query Processing for Uncertain Data. In *Proceedings of the International Conference on Information and Knowledge Managemen, CIKM*, 2010.

[7] M. E. Khalefa, M. F. Mokbel, and J. J. Levandoski. PrefJoin: An Efficient Preference-Aware Join Operator. In *Proceedings of the International Conference on Data Engineering, ICDE*, 2011.

[8] W. Kießling. Foundations of Preferences in Database Systems. In *Proceedings of the International Conference on Very Large Data Bases, VLDB*, 2002.

[9] J. J. Levandoski, M. F. Mokbel, and M. E. Khalefa. FlexPref: A Framework for Extensible Preference Evaluation in Database Systems. In *Proceedings of the International Conference on Data Engineering, ICDE*, 2010.

[10] J. J. Levandoski, M. F. Mokbel, and M. E. Khalefa. Preference Query Evaluation over Expensive Attributes. In *Proceedings of the International Conference on Information and Knowledge Managemen, CIKM*, 2010.

[11] Microsoft MapPoint: http://www.microsoft.com/mappoint/.

[12] PostgreSQL: http://www.postgresql.org.

[13] M. L. Yiu and N. Mamoulis. Efficient Processing of Top-k Dominating Queries on Multi-Dimensional Data. In *Proceedings of the International Conference on Very Large Data Bases, VLDB*, 2007.

Context Modelling and Context Awareness: steps forward in the Context-ADDICT project*

Cristiana Bolchini and Giorgio Orsi and Elisa Quintarelli and Fabio A. Schreiber
and Letizia Tanca
Dipartimento di Elettronica e Infomazione - Politecnico di Milano
Piazza Leonardo da Vinci, 32 - 20133 Milano (Italy)
{bolchini,quintare,schreibe,tanca}@elet.polimi.it, giorgio.orsi@cs.ox.ac.uk

Abstract

We give an account of the researches on context-aware information tailoring which are going on within the PEDiGREE[1] group at Politecnico di Milano, starting from a foundational framework for the life-cycle of context-aware information systems, in which the system design and management activities consider context as an orthogonal, first-class citizen. The design-time and run-time activities involved in this life-cycle provide material for stimulating research, summarized in this paper.

1 Introduction

In a world of global networking, the increasing amount of heterogeneous information, available through a variety of channels, has made it difficult for users to find the right information at the right time and at the right level of detail. This is true not only when accessing information from portable, mobile devices, characterized by limited – although growing – resources and by high connection costs, but also when using powerful systems, since the amount of "out-of-context" answers to a given user request may be overwhelming. Therefore, we evolve from the need of personalizing information presentation to the difficult task of filtering and personalizing the information itself.

User tastes and profile, external environmental factors, current trends and involved phenomena are today recognized as parts of the notion of context [9, 30, 1, 6], enriching the initial concept only based on time and location. In the *cognitive-science view*, "context is used to model interactions and situations in a world of infinite breadth, and human behaviour is key in extracting a model"; in our work we focus on the less ambitious *engineering view*, where "context is useful for representing and reasoning about a restricted state space within which a problem can be solved" [9], and use it as the basis for information personalization and filtering.

Modern applications adopt a context-aware perspective to manage: i) *communication* among users and among systems [11], or between the system and the user [10, 14]; ii) *situation-awareness*, like modelling location and environment aspects (physical situation) [25, 22] or the current user activity (personal situation) [18];

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

*This research has been partially funded by the European Commission, Programme IDEASERC, Project 227977-SMScom. Giorgio Orsi is supported by the Oxford Martin School - Institute for the Future of Computing.

[1] PErvasive Database GRoup of EnginEers

iii) *knowledge chunks*: determining the set of situation-relevant information [27, 7], services [26, 16] or behaviours [4, 31].

Because the applications are so different, we feel that context modelling should be addressed independently of the specific objective, therefore on the side of the operational system we design the *context management system*, where all the above-mentioned variables contributing to context, rather that being considered just as all the other system parameters (*holistic* view), have the special role of *context variables* and are modelled orthogonally with respect to the other instantaneous system inputs. In the Context-ADDICT project [7] context is hierarchically modelled in terms of *observable* parameters that have a symbolic internal representation within a *context schema*[2], and some of which correspond to numerical values gathered from the environment by means of suitable appliances like sensors or RFID tags. At run time, the context is "sensed", and then validated, when the discovered combinations of values constituting the current context are verified against the context schema. This triggers a context-aware behaviour: delivery of context-aware data or actuation of context-aware operations.

This general view of the design-time and run-time activities involved in context-awareness encompasses all kinds of context-aware systems, however in Context-ADDICT we concentrate on a data-oriented perspective, where users are presented with the right information at the right moment, while all the data that are not really interesting for the current context are considered as "noise" and thus removed. This process is referred to as *context-aware information tailoring*.

The activities involved in context-aware information tailoring provide material for stimulating research, summarized with the help of Figure 1, which conceptually illustrates the vision adopted in Context-ADDICT to support all the phases of the tailoring process. We consider a general scenario where several classes of users access, through a (set of) application(s), a variety of information, made available by internal or external data sources, from sensors used for monitoring the environment to datasets of any format. The knowledge of the context is used to control the flow of information reaching the users, excluding the *information noise*. A fundamental aspect of the whole context management architecture is to keep the elaborations, queries and data retrieval operations necessary to present the context-tailored information transparent to the user, independently of the type of information source. The core of the architecture consists of the following main components:

a) **Design-Time Context Manager**: in charge of supporting i) the design of the context schema of the specific application scenario by means of a suitable *context model*, and ii) for each context admitted by the context schema, its association with the appropriate *context-aware view* over the available data sources.

b) **Context-Aware Personalization Manager**: it determines, at each instant, the currently *active context*, applies contextualization to the queries issued towards the different data sources and delivers the context-aware data to the users and applications. Some of the aspects characterizing the current context can be automatically derived (such as the location and time of the user, her/his role within the application scenario, the kind of device used to access the data), while others may require an explicit specification by the user (for instance, her/his current topics of interests).

c) **Data Access Manager**: it offers access to data that can reside internally, for instance in a database, or in other, external data sources, providing an early-tailored, context-aware unified answer.

The components we have just described constitute the basic architecture of the Context-ADDICT project. While striving to design this architecture in detail and to adopt it in some real-life cases, we came across several of the research challenges that are currently absorbing our group. Let us consider the **Design-Time Context Manager**; here, the designer's role is to envisage the possible contexts the user will incur during the system's life. However the application requirements of a context-aware system are intrinsically dynamic and thus can evolve, therefore it is highly probable that, after some time, the context perspectives and parameters anticipated at design time be not applicable any more. Accordingly, the context schema used as the basis of the tailoring

[2] A context schema specifies the possible contexts for the current application domain.

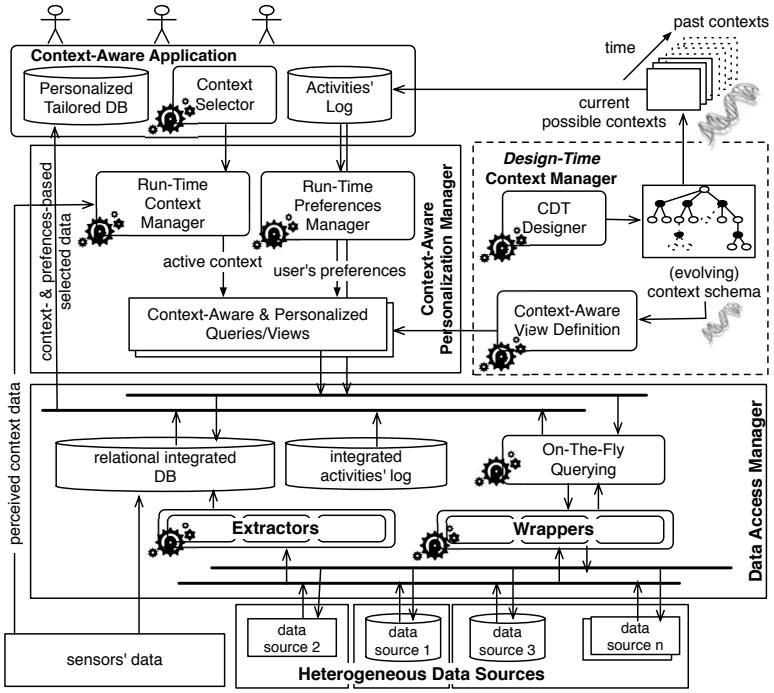

Figure 1: A general view of the architecture of a data-oriented, context-aware system.

process should be allowed to change over time, along with the design and assignment of the related context-aware views. These two strictly related problems originated the research that is briefly described in Section 2. More about context-aware query tailoring will be described also in Section 4.

Note that the user interests and preferences may vary according to the context the user is currently in, for a change in context may change the relative importance of information. In the **Context-Aware Personalization Manager**, *contextual preferences* are used to refine the views associated with contexts, by imposing a ranking on the data of a context-aware view and adding the opportunity to dispose of the less interesting portion if needed. On the other hand, since we cannot actually expect a user to manually specify the long list of preferences that might be applied to all available data when a context becomes active, we propose a methodology and a system, PREMINE, where data mining is used to learn contextual preferences from the previous user's querying activity. This research is briefly described in Section 3.

The proliferation of freely-accessible data-intensive websites, as well as initiatives for open-accessible linked data in the Web [5], provide the users with potential sources of precious information. The **Data-Access Manager**, discussed in Section 4, supports the context-aware querying of heterogeneous and dynamic data sources, retrieving from the sources only the data that are consistent with the current context. Among the information sources we also consider devices which are spatially distributed and possibly mobile, the task of some of which is to monitor different kinds of physical phenomena for application support. Still at the data-access level, we designed the PerLa language and middleware which allow for declarative gathering of data from the environment, where some of these "pervasive" data are used to provide information to the operational system, while a context-management extension to the same language permits to declare the context and to provide the actual values for the context parameters.

2 Design-Time Context Manager

A key element in the design of context-aware systems is the representation and management of context and of its attributes, to be able to define a relationship between each context and the aspects of interest (such as relevant information, applicable "rules" and behaviour, presentation, etc.) of the application scenario. As all well-conceived formalisms, a model that allows the designer to specify such aspects should satisfy at least the requirements of generality (it should be suited to represent any application scenario), modularity (describing the context perspectives at the correct level of granularity), readability (to serve as design documentation), expressivity (allowing querying, reasoning or constraint specification on the contexts of a given target application) [6, 22].

We devoted our initial efforts to the definition of a suitable context model, the *Context Dimension Tree* formally presented in [7], used to specify at design-time the context schema, that is, all possible contexts the user will be acting in. The *context schema* (or *CDT*) of a given scenario is a hierarchical structure consisting of i) *context dimensions* (black nodes), modeling the different perspectives through which the user perceives the application domain and ii) the allowed *dimension values* (white nodes), i.e. the actualized parameters based on which the context-aware information, behaviours, etc. are to be selected. Each context is then defined as a combination of *context element*s, that is, $<$*dimension-value*$>$ pairs. The adoption of a hierarchical organization allows us to employ different abstraction levels to specify and represent contexts. It is also possible to define appropriate constraints on the schema, that prevent some of the combinations of context elements, if meaningless for the current application. For example, consider the university campus scenario where users are students of any level (undergraduate, graduate, PhD), faculty members and guests. The university provides a mobile application to allow an easy, context-aware, personalized access to all information that might be of interest, like courses, schedules, rooms, other activities, local transportation, dining resources. Here, a valid *context* is the one of a student, who is interested in finding information about courses, formally expressed as $user = student \wedge interest\_topic = courses$, while the context $user = undergraduate \wedge interest\_topic = research$ might be considered meaningless by the designer and thus ruled out by writing an appropriate constraint.

It is easy to see that the burden of explicitly defining the portion of data deemed interesting for each context is a time-consuming task given the possibly high number of valid contexts; therefore the naive *per context* association of views to contexts evolved towards the definition of a methodology working on a *per context-element* basis, which supports the designer in the definition of a view for each context element – called *relevant area*, or $\mathcal{R}el$ –; based on this specification, the system automatically derives the data portion associated with a context by opportunely combining the relevant areas of the involved context elements. The advantage is twofold: i) to reduce the designer's work and ii) to achieve a high flexibility w.r.t. an evolving application scenario. The approach is fully general, and has been profitably applied to other data models; in particular, in order to add the necessary reasoning capabilities, we have explored the representation of the context model by means of ontologies [23] and logic programs [24]. For data tailoring in the relational model [8], we have defined operators that apply to sets of relations and support the composition of context-aware views. As a result, the designer's effort is devoted to the definition of the CDT and of the $\mathcal{R}el$s, while a tool[3] performs, for each context, the automatic computation of the resulting context-aware views.

Starting from techniques devised in the literature on schema evolution in various fields [2, 19], our recent research proposes strategies to flexibly manage the evolution of the context schema. During system operation, the initial vision of the designer about the possible contexts pertaining to a given application scenario may change, and new dimensions or values might come into play or old ones might become not applicable. As a result, the context schema has to change, and for each new node the associated view needs to be computed or a modification of some existing views might be necessary. For instance, in the university scenario, a change from quarters to semesters (i.e., a change of dimension values) would cause a revision of the schema and the need to

[3] The activities of the Design-Time Context Manager are supported by *CADDFrame*, integrating a CDT Designer to define the CDT, a tool to specify the $\mathcal{R}el$s, and an engine that combines them to compute the final views associated with each valid context.

write new views, associated with the semester data. A context schema change might also be triggered when the user device detects some environmental parameter values that are not envisaged by the current schema.

The Design-Time Context Designer supports context schema modifications by allowing the use of a predefined set of *evolution operators* that keep the coherence between the schema changes and the (induced) changes in the encompassed contexts. The sequence of the modifications is logged, and used to refresh the user context and views in such a way that, when the user device detects a context change and requests the related data, these are delivered consistently with the new schema. This is made possible by the fact that the per-element view-construction strategy is employed, and thus only a limited number of views – those related to the nodes that are involved in the evolution operation – have to be added or revised by the designer.

3 Context-Aware Personalization Manager

Data personalization based only on the notion of context represents but a partial solution to the problem of information reduction, since the tailoring turns out to be often too coarse-grained. For example, consider the context of Bob, a student interested in finding a free room for the afternoon; a contextual system will suggest only the rooms close to his current location, and will not be able to propose any ranking or further filtering of the contextual data according to Bob's tastes. Actually, the context often plays a crucial role in determining different prefences of the same person: for instance Bob could be very interested in quiet rooms when he is alone, and in open spaces when he is with his friends.

To obtain a more effective personalization, the *Run-Time Preference Manager* (see Figure 1) couples the knowledge of context with the user personal preferences: this allows to specify which information represents the actual interests and needs for Bob, when he is in a particular context (alone or with friends). In this direction, some approaches [29, 21, 17, 20] have been proposed for personalizing relational data (both tuples and attributes) on the basis of contextual preferences defined by the user. However, when considering a large variety of data and a rich set of possible contexts, the manual specification of an extensive list of preferences may be onerous for the user, who may be discouraged w.r.t. the activity of explicitly indicating his/her preferences. A way around this problem is to learn the user tastes from a log of his/her past behaviour, i.e., the past data-querying activities. We use data-mining algorithms to discover implicit contextual behaviours, in the form of *association rules* that correlate each contexts with the values accessed in it. In particular, a σ-rule $\overline{r}_\sigma$ on the relation schema $R(X)$ is a triple $\langle C \rightarrow cond, sup, conf \rangle$, where $C \rightarrow cond$ is an association rule, C is a context and *cond* a conjunction of simple conditions in the form $A = value$, with $A \in X_i$ for some $R_i(X_i) \in R(X)$; sup and $conf$ indicate the support and the confidence of $C \rightarrow cond$, respectively. The support corresponds to the frequency of $C \wedge cond$ be true in the log storing the user activities; the confidence corresponds to the (conditional) probability of finding *cond* true in the log of the activities done in context C, and is given by $\frac{sup(C \wedge cond)}{sup(cond)}$. For example, if we consider the table CLASSROOM(NAME,BUILDING,FLOOR,LOCATION,TYPE) in the dataset of our scenario, and we mine the σ-rule $\langle user = student \wedge situation = alone \rightarrow \sigma_{type=computerized} CLASSROOM, 0.8, 0.9 \rangle$ by analysing Bob's activities, we can state that, when alone, he is often interested in computer-equipped classrooms (with a support 0.8 and a confidence 0.9).

σ-rules are used to learn preferences on tuples, but the preferences can be mined also on attributes. A π-rule $\overline{r}_\sigma$ on the relation schema $R(X)$ is a triple $\langle C \rightarrow A, sup, conf \rangle$, where C is a context and $A \in X_i$ for some $R_i(X_i) \in R(X)$. For example, the π-rule $\langle user = student \wedge situation = alone \rightarrow \{location\}, 0.7, 0.8 \rangle$ states that Bob, when alone, often visualizes the classroom location.

Our approach, differently from the majority of recommendation systems, does not require any explicit input from the user about his/her preferences; we are studying its seamless combination with recommendations that can be used to personalize the views for new users, when the history of their activities is not available yet, on the basis of the behaviours of other, "similar" users in the same context.

4 Data-Access Manager

In Context-ADDICT the data sources are generally *dynamic*, *transient*, and *heterogeneous* in both their data models (e.g., relational, XML, RDF) and schemas[4]. Therefore, in order to access their content, it is first necessary to understand their schemas and to represent them in a suitable semantic formalism (e.g., description logics or Datalog-like languages) to enable their semi-automatic reconciliation. We designed *extractors* (Figure 1) to automatically derive, through reverse-engineering, the *external schemas* from the schemas of the data sources. In particular the extractors are *domain-aware*, i.e., they include in the external schema only the data-source entities which have a counterpart in a global schema [12] which is used as a uniform representation of the available information in the integration system. Each external schema is then (semi-)automatically mapped to the global schema, which can be eventually queried using a suitable query language, e.g. SQL or SPARQL, depending on the model chosen for the global schema. The mappings between the external schemas and the global schema are provided in a semi-automatic way, using schema matching tools [3].

In Context-ADDICT, the schemas are expressed using the CA-$\mathcal{DL}$ data language that allows to: (i) uniformly represent the data and the user context(s) in any application domain and (ii) support efficient context-aware query distribution and answering, for external data-sources context-based information filtering is performed at run-time (*early tailoring*), by selecting fragments of the global schema that are considered relevant for the current context. These relevant areas are computed by means of the per-context-element strategy discussed in Section 2. Each relevant area is matched against the external schemas produced by the extractors, inducing, through the mappings, corresponding context-aware fragments of the external schemas [23]. Data access proceeds as follows: the queries formulated over the global schema are first translated into context-aware queries using the specifications provided by the relevant areas. Then, the *contextualized queries* are reformulated in terms of the global schema and of the mappings, producing queries over the data sources. The queries are then distributed to the corresponding data sources and finally translated into the native language of each target source by means of suitable wrappers.

A similar perspective is adopted to access data coming from sensors. We developed PerLa[5](PERvasive LAnguage) [28] a declarative, high-level language that allows to query a pervasive system within the unifying setting provided by Context-ADDICT, hiding the difficulties related to the need of handling different technologies. A database-like abstraction of the whole sensor network is provided in order to hide the high complexity of low-level programming, allowing users to retrieve both functional and non-functional data from the system and send configuration/activation commands to the sensors in a fast and easy way. A middleware provides an abstraction for each device in terms of a *logical object*, and supports the execution of PerLa queries. During the design of the middleware, we strove to make the definition and the addition of new devices easier by minimising the amount of low-level code the user has to write to make the new device recognizable by the system.

Context awareness has a two-fold role with respect to sensors' data: part of the collected information contributes to the information base to be made available through the application and is filtered according to context as in the general case; part of it must be directly used to determine the context itself, such as the time of the day or the location (e.g., from GPSs). In the latter situation, the information is directly exploited by the *Run-Time Context Manager* (see Figure 1) to actualise the correct context. PerLa, originally conceived only for querying sensors, has thus been extended with statements to i) *define the CDT structure*, also with the capability of acquiring that part of context information that cannot be deduced from sensor readings; ii) *create a context* on a defined CDT; iii) *activate/deactivate a context* at run-time as by the actual values of the context variables; iv) *perform the contextual actions* required on the system, e.g.: activating actuators, changing measurement modalities, cutting and tailoring queries from a general query stereotype.

[4] In our research we mostly concentrate on data sources whose schema is available, while for less structured data sources we resort to techniques found in state-of-the-art literature [15].

[5] http://perlawsn.sourceforge.net/index.php

5 Conclusions

We have given a brief report of our understanding of the research on context-awareness, and of the activities on context-aware information systems going on within the PEDiGREE group at Politecnico di Milano. The overall research area is so stimulating that we continuously encounter topics that should be investigated, like context sharing, automatic synthesis (from observing the environment) of the possible contexts and of the views to be associated with them, automatic recognition of the next active context to the end of anticipating critical situations [13], uncertainty and inconsistency of the context data, stability of the system when the context changes become too frequent for adaptation, support of non-functional requirements like context-aware data quality [4], and many others, which can be a stimulus to the scientists who are attracted by this ever-growing research area.

References

[1] M. Baldauf, S. Dustdar, and F. Rosenberg. A survey on context-aware systems. *Intl J. of Ad Hoc Ubiquitous Computing*, 2(4):263–277, 2007.

[2] A. Balmin, Y. Papakonstantinou, and V. Vianu. Incremental validation of XML documents. *ACM Trans. on Database Systems*, 29(4):710–751, 2004.

[3] Z. Bellahsene, A. Bonifati, and E. Rahm. *Schema Matching and Mapping*. Springer-Verlag, Heidelberg (DE), 2011.

[4] L. Bertossi, F. Rizzolo, and L. Jiang. Data quality is context dependent. To appear in *Proc. of the 4th Intl Work. on Business Intelligence for the Real Time Enterprise*, 2010.

[5] C. Bizer, T. Heath, and T. Berners-Lee. Linked data - the story so far. *Int. J. Semantic Web Inf. Syst.*, 5(3):1–22, 2009.

[6] C. Bolchini, C. A. Curino, E. Quintarelli, F. A. Schreiber, and L. Tanca. A data-oriented survey of context models. *SIGMOD Record*, 36(4):19–26, 2007.

[7] C. Bolchini, C. A. Curino, E. Quintarelli, F. A. Schreiber, and L. Tanca. Context information for knowledge reshaping. *Intl J. of Web Engineering and Technology*, 5(1):88–103, 2009.

[8] C. Bolchini, E. Quintarelli, and R. Rossato. Relational data tailoring through view composition. In *Proc. 26th Intl Conf. on Conceptual Modeling, ER*, pages 149–164, 2007.

[9] P. Brézillon and S. Abu-Hakima. Using knowledge in its context. *AI Magazine*, 16(1):87–91, 1995.

[10] S. Buchholz, T. Hamann, and G. Hübsch. Comprehensive structured context profiles (CSCP): Design and experiences. In *Proc. of 1st Intl Work. on Context Modelling and Reasoning*, pages 43–47, 2004.

[11] H. Chen, T. Finin, and A. Joshi. An intelligent broker for context-aware systems. In *Proc. of Intl Conf on Ubiquitous Computing - Poster Session*, pages 183–184, 2003.

[12] C. A. Curino, G. Orsi, E. Panigati, and L. Tanca. Accessing and documenting relational databases through OWL ontologies. In *Proc. of 8th Intl Conf. on Flexible Query Answering Systems*, pages 431–442, 2009.

[13] Y. Ding, H.R. Schmidtke, and M. Beigl. Beyond context-awareness: context prediction in an industrial application. In *Proc. of the 12th Intl Conf. on Ubiquitous Computing*, pages 401–402, 2010.

[14] Barbara Pernici (ed.). *Mobile Information Systems - Infrastructure and Design for Adaptivity and Flexibility*. Springer-Verlag, Heidelberg (DE), 2006.

[15] G. Gottlob, C. Koch, R. Baumgartner, M. Herzog, and S. Flesca. The Lixto data extraction project - back and forth between theory and practice. In *Proc. of the 23rd Symp. on Principles of Database Systems*, pages 1–12, 2004.

[16] T. Gu, H. K. Pung, and D. Q. Zhang. A service-oriented middleware for building context-aware services. *J. of Network and Computer Applications*, 28(1):1–18, 2005.

[17] E. Jembere, M. O. Adigun, and S. S. Xulu. Mining context-based user preferences for m-services applications. *Web Intelligence*, pages 757–763, 2007.

[18] M. Kaenampornpan and E. O'Neill. An integrated context model: Bringing activity to context. In *Proc. of Work. on Advanced Context Modelling, Reasoning and Management*, 2004.

[19] M. M. Lehman. Software's future: Managing evolution. *IEEE Software*, 15(1):40–44, 1998.

[20] J. J. Levandoski, M. F. Mokbel, and M. E. Khalefa. Caredb: A context and preference-aware location-based database system. *PVLDB*, 3(2):1529–1532, 2010.

[21] A. Miele, E. Quintarelli, and L. Tanca. A methodology for preference-based personalization of contextual data. In *Proc. of the 12th Intl Conf. on Extending Database Technology*, pages 287–298, 2009.

[22] A. Mileo, D. Merico, and R. Bisiani. Support for context-aware monitoring in home healthcare. In *Intelligent Environments (Workshops)*, pages 177–184, 2009.

[23] G. Orsi. *Context Based Querying of Dynamic and Heterogeneous Information Sources*. PhD thesis, Politecnico di Milano, 2011.

[24] G. Orsi and L. Tanca. Context modelling and context-aware querying: can datalog be of help? To appear in *Proc. of the Datalog 2.0 Workshop*, 2010.

[25] D. Preuveneers, J. van den Bergh, D. Wagelaar, A. Georges, P. Rigole, T. Clerckx, E. Berbers, K. Coninx, and K. de Bosschere. Towards an extensible context ontology for ambient intelligence. In *Proc. of the 2nd European Symp. on Ambient Intelligence*, pages 148–159, 2004.

[26] P.G. Raverdy, O. Riva, A. de La Chapelle, R. Chibout, and V. Issarny. Efficient context-aware service discovery in multi-protocol pervasive environments. In *Proc. of 7th Intl Conf. on Mobile Data Management*, pages 3–11, 2006.

[27] Y. Roussos, Y. Stavrakas, and V. Pavlaki. Towards a context-aware relational model. In *Proc. of 1st Intl Context Representation and Reasoning Work.*, pages 7.1–7.12, 2005.

[28] F. A. Schreiber, R. Camplani, M. Fortunato, M. Marelli, and G. Rota. Perla: A language and middleware architecture for data management and integration in pervasive information systems. To appear in *IEEE Trans. on Software Engineering*, 2011.

[29] K. Stefanidis, E. Pitoura, and P. Vassiliadis. Adding context to preferences. In *Proc. Intl Conf. Data Engineering*, pages 846–855, 2007.

[30] T. Strang and C. Linnhoff-Popien. A context modeling survey. In *Intl Work. on Advanced Context Modelling, Reasoning and Management*, pages 31–41, 2004.

[31] M. Theodorakis, A. Analyti, P. Constantopoulos, and N. Spyratos. A theory of contexts in information bases. *Information Systems*, 27(3):151–191, 2002.

The QueRIE system for Personalized Query Recommendations

Gloria Chatzopoulou[1] Magdalini Eirinaki[2] Suju Koshy[2] Sarika Mittal[2]
Neoklis Polyzotis[3] Jothi Swarubini Vindhiya Varman[2]

[1] Vanderbilt University [2] San Jose State University [3] UC Santa Cruz

Abstract

Interactive database exploration is a key task in information mining. However, users who lack SQL expertise or familiarity with the database schema face great difficulties in performing this task. To aid these users, we developed the QueRIE system for personalized query recommendations. QueRIE continuously monitors the user's querying behavior and finds matching patterns in the system's query log, in an attempt to identify previous users with similar information needs. Subsequently, QueRIE uses these "similar" users and their queries to recommend queries that the current user may find interesting. We discuss the key components of QueRIE and describe empirical results based on actual user traces with the Sky Server database.

1 Introduction

Database systems are becoming increasingly popular in the scientific community, as tools to access and analyze large volumes of scientific data. Prominent examples include the Genome browser[1] that hosts a genomic database, and SkyServer[2] that stores large volumes of astronomical measurements. Such databases are typically accessed through a web interface that allows users to pose queries through forms or in some declarative query language (e.g., SkyServer users can submit SQL queries directly).

Despite the availability of querying tools, users of these systems may still find it challenging to discover interesting information. Specifically, users may not know which parts of the database hold useful information, may overlook queries that retrieve relevant data or might not have the required expertise to formulate such queries. An exhaustive exploration of the database is also practically impossible, due to the continuously growing size of the data. To address this issue, we designed the QueRIE[3] system that assists users in the interactive exploration of a large database. The core idea is to present a user with personalized query recommendations, which are relevant to the user's information needs and can serve as "templates" for query formulation. The user is able to directly submit or further refine these queries, instead of having to compose new ones.

QueRIE is built on a simple premise that is inspired by Web recommender systems: If a user *A* has similar querying behavior to user *B*, then they are likely interested in retrieving the same data. Hence, the queries of

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

[1] `http://genome.ucsc.edu/`
[2] `http://cas.sdss.org/`
[3] Query Recommendations for Interactive data Exploration

user *B* can serve as a guide for user *A*. One obvious approach to realize this idea is to leverage well known collaborative filtering techniques, which were popularized in Web recommender systems. However, this transfer introduces several technical challenges. First, the use of declarative queries implies that users may retrieve the same data through queries that are syntactically different. This complicates greatly the computation of user similarity, since it is not trivial to simply compare the corresponding queries – we essentially have to solve the notoriously difficult query-equivalence problem. A second challenge is the absence of an explicit rating system for the queries posed by a user – how do we know which queries are important in the computation of user similarity? Finally, it is important to recommend queries that are intuitive, i.e., queries that the user can understand and refine if necessary. A user may not find a recommendation useful if the query is too "synthetic".

QueRIE addresses these challenges by employing a closed-loop approach. Specifically, QueRIE decomposes each query into basic elements that capture the essence of the query's logic. These elements are used to compute similarities between users, and also to compute a signature of the user's querying behavior, and to some extent of the user's information needs. Recommendations are generated by mining queries from the system log that match well with the signature. Hence, the user is presented with queries that match her querying behavior, and that are likely to be more intuitive than purely synthetic queries.

2 Related Work

Even though the problem of generating personalized recommendations has been broadly addressed in the Web context [11], only a handful of related works exist in the database context. Some work has been done in the area of personalized recommendations for keyword or free-form query interfaces [12]. In this scenario, a user queries a database using keywords or free-form text and the personalization system recommends items of interest. Our approach is different from this scenario because it aims to assist users who query relational databases using either ad-hoc or form-based queries. Also, our framework recommends queries instead of "items" from the database. Finally, QueRIE does not require from the users to explicitly declare their preferences beforehand in order to generate recommendations.

A multidimensional query recommendation system is proposed in [6, 7]. In this work the authors address the related problem of generating recommendations for data warehouses and OLAP systems. In this work, the authors propose a framework for generating OLAP query recommendations for the users of a data warehouse. Although this work has some similarities to ours (for example, the challenges that need to be addressed because of the database context), the techniques and the algorithms employed in the multidimensional scenario (for example, the similarity metrics and the ranking algorithms) are very different to the ones we propose.

The necessity of a query recommendation framework is emphasized in [9], where the authors outline the architecture of a collaborative query management system targeted at large-scale, shared-data environments. As part of this architecture, they suggest that data mining techniques can be applied to the query logs in order to generate query suggestions. The authors present a general outline of a framework for query recommendations pointing out that this is a challenging process. However, they do not provide any technical details on how such a recommendation system could be implemented.

Two recent works propose frameworks for query recommendations using the information recorded in the query logs [13, 8]. In [13], the authors propose a query recommender system that represents the past queries using the most frequently appearing tuple values. Then, after predicting which new tuples might be of interest to the end user, they reconstruct the query that retrieves them. This approach works better with relations that have discrete attribute values, contrary to scientific databases, where most attributes are numeric. The authors also propose a global ranking of the queries, based on the statistics of the database and not the query logs. Both approaches are evaluated in a preliminary empirical study, yet no discussion on scalability issues is provided. In [8] the authors propose SnipSuggest, a system meant to assist users when formulating SQL queries. Using the information in query logs, the system generates a DAG representing the relationships between different clauses

in the queries. Based on the current user's query, the system ranks the outgoing edges and recommends the nodes/clauses of the most similar ones. This work differs in ours in several ways. SnipSuggest recommends possible additions to various clauses in the current user's query, and not complete queries. Moreover each query is treated independently of any previous one, even if they belong to the same user session.

3 An Abstract Framework for Query Recommendations

This section discusses an abstract framework for generating query recommendations. The abstract framework is essentially a workflow that takes as input the queries of previous users and the queries of a current user, and outputs query recommendations. We discuss two possible instantiations of the workflow in the next section.

We consider a setting where users explore a relational database through SQL queries. Specifically, each user interacts with the database in the form of *sessions*, where each session consists of a sequence of SQL queries. We expect the queries in the same session to be correlated, as we assume that the user is trying to mine useful information from the data in an exploratory fashion, i.e., the formulation of the next query in the session is highly dependent on the results of previous queries. To simplify our presentation, we assume that each user has a single session with the database. This assumption can be lifted in a straightforward manner at the expense of more complicated notation.

Given a user i, let Q_i denote the set of queries that the user has posed thus far in a single session. We introduce the notion of a *session summary* to summarize the characteristics of the queries posed in the session. Contrary to Web recommender systems, where a session summary can be readily constructed as the items that a user visits/rates/purchases, there exist several ways to model a session of SQL queries. For instance, a crude summary may contain the names of the relations that appear in the queries of the user, and the importance of each relation can be measured as the number of queries that reference it. On the other extreme, a detailed summary may contain the actual results inspected by the user, along with an explicit rating of each result tuple. The different possibilities represent trade-offs between detail and conciseness, and as we will see later, also affect the quality of the generated queries. In what follows, we use S_i to represent the session summary for user i. User $i = 0$ will always represent the current user (for whom recommendations are generated), whereas $i = 1, \ldots, n$ represents past users of the system. In a slight abuse of notation, we use S_i to represent both the session summary and user i.

Our conceptual framework generates recommendations for S_0 in two steps. First, the framework computes a "predicted" summary S^{pred} that captures the importance of different query characteristics for user S_0 . Note that S^{pred} may contain characteristics that already appear in the queries of S_0, but also new ones that the user has not used yet. The summary S^{pred} is used in the second step of the recommendation workflow, as the "seed" to generate recommendations. We further detail the two steps below.

The predicted summary is computed as $S^{\text{pred}} = f(\alpha, S_0, S_1, ..., S_n)$, where f is a function that combines information from summaries $S_0, S_1, \ldots, S_n$. Clearly, the function will depend on the specific model for session summaries. Parameter α is a mixing factor, which controls the importance of S_0 (the current user's session) with respect to $S_1, \ldots, S_n$ (the sessions of other users). Recall that S^{pred} includes query characteristics that already appear in the queries of S_0. The reason is that we want to recommend queries that extend or restructure the queries already posed by S_0, in addition to completely different (yet relevant) queries. By setting $\alpha = 0$, we ignore completely the queries in S_0, whereas $\alpha = 1$ has the exact opposite effect, i.e., only the queries in S_0 affect the recommendations. We have found in our experiments that neither of these two extremes are satisfactory, which justifies the introduction of α in our framework. It is interesting to contrast our approach to Web recommender systems, where the equivalent of S^{pred} is computed using $\alpha = 0$, i.e., based solely on information from other users. (As an example, it is not meaningful to recommend to a user a movie that they have already watched.)

Summary S^{pred} is used to generate the queries that are presented to the user as recommendations. One

approach is to synthesize queries based on the characteristics present in S^{pred}, but this is not likely to work well for several reasons. First, it is important to recommend queries that have non-empty result sets, and hence it will be necessary to verify this property for every candidate recommendation. Second, it is extremely difficult to synthesize intuitive queries that combine specific characteristics, unless there is some semantic knowledge about the data and/or the schema (which is difficult to acquire in itself). Our approach is completely different: we mine the query log of the DBMS to select queries that have a good match to the characteristics described in S^{pred}. The recommendations are likely to be intuitive and easy to understand, since they correspond to queries formulated by other human users. Furthermore, we can verify easily that these queries return non-empty results, by examining the metadata in the system's query log[4].

Overall, our framework consists of the following components: (a) a model for session summaries, (b) a method to compute the session summaries $S_0, \ldots, S_n$, (c) a method to compute S^{pred}, and (d) a method to select queries based on S^{pred}. An interesting feature is that the framework forms a closed-loop, receiving SQL queries as input and generating SQL queries as output. This design follows from our starting assumption that users interact with the database through a declarative query language.

4 Tuple-based and Fragment-based Query Recommendations

We now describe two instantiations of the abstract recommendation framework, namely a *tuple-based* recommendation engine and a *fragment-based* recommendation engine. As discussed below, the two instantiations represent a trade-off between efficiency and quality of recommendations.

Tuple-Based Recommendation Engine. The session summary S_i is represented as a weighted vector, where every coordinate corresponds to a distinct database tuple. The weight $S_i[\tau]$ represents the importance of a given tuple τ in the session of user i, and is non-zero only if τ is a witness for at least one query in the session. The intuition is that S_i captures the tuples in the base tables that are touched by the queries in the session. Hence, sessions that contain equivalent queries will map to the same summary.

We consider two schemes for setting $S_i[\tau]$ for a witness τ: a frequency-based scheme, where $S_i[\tau]$ essentially corresponds to the number of queries that have τ as the witness; and a result-based scheme, where $S_i[\tau]$ is inversely proportional to the result sizes of the queries for which τ is a witness. The idea behind the last scheme is similar to the IDF concept from information retrieval–τ obtains a higher weight if it is a witness to queries with small results.

We compute S^{pred} as follows: $S^{\text{pred}} = \alpha \cdot S_0 + (1-\alpha) \cdot \sum_{i=1,\ldots,n} sim(S_i, S_0) \cdot S_i$, where $sim(S_i, S_0)$ is a similarity metric between the two vectors (e.g., cosine similarity). This approach is inspired by Web recommender systems, where the idea is to bias the recommendations based on users that exhibit similar behavior to the current user. The difference is that we use the mix-in factor α to blend in the behavior of the current user. Overall, S^{pred} yields a weight per tuple that corresponds to the importance of the tuple to the user's exploration.

Having computed S^{pred}, we output as recommendations the queries of past users whose witnesses match the high-weight tuples in S^{pred}. Specifically, for each candidate query Q (we maintain a sample of past queries as our candidate pool), we compute the similarity $sim(S_Q, S^{\text{pred}})$, where S_Q is the summary of a session that comprises solely query Q. The few candidate queries with the highest similarity metric are returned as the recommendation to the user. (The number of returned queries is a parameter of the framework.)

Overall, the tuple-based approach captures the user's querying behavior at a very fine level of detail–the individual witnesses to the user's queries. Moreover, it handles readily the issue of equivalent declarative queries, since the underlying witness sets are exactly the same. The downside is the increased complexity, since, in principle, the session summaries grow linearly with the size of the database. Fortunately, it is possible to implement

[4]One obvious concern is that a query may not return any results if the database has been updated. However, this scenario is rather unlikely for scientific data sets, which are typically append-only.

this method efficiently by employing randomized sketching techniques (e.g., AMS sketches [3] or min-hash sketches [5]) to compress the summaries and compute the similarity metrics with a small loss in precision.

Fragment-Based Recommendation Engine. The fragment-based approach works similarly, except that the coordinates of the session summaries correspond to fragments of queries instead of witnesses. We identify as fragments the following syntactical features of the queries in the session: attribute references, tables references, join and selection predicates. At a high level, the idea behind this approach is to recommend queries whose syntactical features match the queries of the current user.

Formally, a session summary S_i is a vector whose cell $S_i[\phi]$ contains a non-zero weight if the fragment ϕ appears in at least one query of the session. Conceptually, the length of the vector is equal to the number of possible fragments, but we expect only few cells to have non-zero values. There are several possibilities for the weighting scheme, and we consider two in our work: binary, which indicates the presence of ϕ, and frequency-based, which assigns to $S_i[\phi]$ the count of queries that contain the fragment.

Under this model, S^{pred} corresponds to a vector that indicates the relevance of query fragments to the current user's exploration. Similar to the tuple-based engine, we compute S^{pred} as a blend of the user's summary S_0 and other users' summaries $S_1, \ldots, S_n$ weighted according to some similarity metric. The difference is that we employ a fragment-fragment approach, which is reminiscent of the item-item paradigm of collaborative filtering systems on the Web. Specifically, we first define a fragment-similarity metric $sim(\phi, \rho)$ that evaluates the similarity of the two fragments in terms of their corresponding weights in the session summaries $S_1, \ldots, S_n$. We use Jaccard's coefficient and cosine similarity for the binary and frequency-based schemes respectively, but our framework can accommodate any metric. Using this metric, each coordinate $S^{\text{pred}}[\phi]$ is computed as a blend of $S_0[\phi]$ and the expression $\sum_{\rho \in R}(S_0[\rho] \cdot sim(\rho, \phi)) / \sum_{\rho \in R} sim(\rho, \phi)$, where R is the set of k fragments (k is a parameter of our framework) that have the highest similarity to ϕ. Intuitively, ϕ obtains a high weight if S_0 contains fragments that co-occur frequently with ϕ in the queries of past users.

The generation of the recommended queries occurs in a similar fashion as the tuple-based approach: each candidate query Q is mapped to a summary S_Q, and the queries that have the highest similarity to S^{pred} are returned as recommendations.

The fragment-based approach clearly captures information at a coarser level of detail, and hence it is expected to miss interesting correlations between users. For instance, two distinct selection predicates will be mapped to different fragments even if they are satisfied by the same tuples in the base tables. The big advantage is that it can be implemented very efficiently, as the space of fragments grows slowly, the summaries are very sparse and the fragment-to-fragment similarities can be computed offline and stored for very fast retrieval when recommendations need to be generated. (This offline preprocessing is not possible in the tuple-based approach, since the similarity metric involves the complete vector S_0 of the current user.)

5 The QueRIE System Prototype

We implemented a prototype of our system that supports the two recommendation engines described in the previous section. The prototype is implemented in Java and runs on top of a standard relational DBMS. The details of our system are described in [10, 1].

Using the prototype, we evaluated the efficacy of our recommendation techniques using real user traces from the SkyServer database. Our results demonstrated that it is possible to generate meaningful query recommendations for a large set of users in the traces. Specifically, our experiments showed that: (a) using $\alpha = 0.5$ produces the best results in terms of recommendation quality, (b) the tuple-based engine generates perfectly precise recommendations (precision is equal to one) for 40% of the users in the test set, (c) the fragment-based engine is noticeably less effective in terms of precision, and (d) the fragment-based engine has much less computational overhead. A detailed discussion of the results can be found in [4, 2]

6 Conclusions and Future Work

Query recommendations can provide valuable guidance for interactive database exploration, particularly for users who lack the expertise to formulate complex queries or who are not familiar with the data. The proposed QueRIE system aims to provide a generic framework where we can develop and evaluate recommendation techniques for this novel domain.

As part of our future work, we intend to investigate a hybrid of the two recommendation engines that achieves a better trade-off between efficiency and precision. We also plan to extend our techniques to form-based query interfaces that are common for Web-accessible databases. Our fragment-based approach is readily applicable for single-form interfaces, but we wish to examine the case of multi-form interfaces where the query is formulated in several steps.

References

[1] J. Akbarnejad, G. Chatzopoulou, M. Eirinaki, S. Koshy, S. Mittal, D. On, N. Polyzotis, and J. Swarubini V. Varman. SQL QueRIE Recommendations (demo paper). In *Proc. of the 36th International Conference on Very Large Data Bases (VLDB '10)*, 2010.

[2] J. Akbarnejad, M. Eirinaki, S. Koshy, D. On, and N. Polyzotis. SQL QueRIE Recommendations: a query fragment-based approach. In *4th International Workshop on Personalized Access, Profile Management, and Context Awareness in Databases (PersDB '10)*, 2010.

[3] N. Alon, Y. Matias, and M. Szegedy. The space complexity of approximating the frequency moments. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing (STOC '96)*, 1996.

[4] G. Chatzopoulou, M. Eirinaki, and N. Polyzotis. Collaborative filtering for interactive database exploration. In *Proc. of the 21st International Conference on Scientific and Statistical Database Management (SSDBM '09)*, 2009.

[5] E. Cohen. Size-estimation framework with applications to transitive closure and reachability. *Journal of Computer and System Sciences*, 55:441–453, 1997.

[6] A. Giacometti, P. Marcel, and E. Negre. Recommending Multidimensional Queries. In *Proc. of the 11th International Conference on Data Warehousing and Knowledge Discovery (DaWaK'09)*, 2009.

[7] A. Giacometti, P. Marcel, E. Negre, and A. Soulet. Query Recommendations for OLAP Discovery Driven Analysis. In *Proc. of the ACM 12th International Workshop on Data Warehousing and OLAP (DOLAP'09)*, 2009.

[8] N. Khoussainova, Y.C. Kwon, M. Balazinska, and D. Suciu. SnipSuggest: Context-Aware Autocompletion for SQL. *PVLDB*, 4(1), 2011.

[9] Nodira Khoussainova, Magdalena Balazinska, Wolfgang Gatterbauer, YongChul Kwon, and Dan Suciu. A case for a collaborative query management system. In *Proc. of the 4th Biennal Conference on Innovative Data Systems Research (CIDR '09)*, 2009.

[10] S. Mittal, J. S. Vindhiya Varman, G. Chatzopoulou, M. Eirinaki, and N. Polyzotis. QueRIE: A Recommender System supporting Interactive Database Exploration (demo paper). In *In Proc. of the 2010 edition of the IEEE International Conference on Data Mining series (ICDM '10)*, 2010.

[11] B. Mobasher. *The Adaptive Web: Methods and Strategies of Web Personalization*, volume 4321 of *LNCS*, chapter Data Mining for Personalization, pages 90–135. Springer, Berlin-Heidelberg, 2007.

[12] A. Simitsis, G. Koutrika, and Y. Ioannidis. Precis: From unstructured keywords as queries to structured databases as answers. *VLDB Journal*, 17(1):117–149, 2008.

[13] Kostas Stefanidis, Marina Drosou, and Evaggelia Pitoura. "You May Also Like" Results in Relational Databases. In *3rd International Workshop on Personalized Access, Profile Management, and Context Awareness in Databases (PersDB '09)*, 2009.

TopRecs$^+$: Pushing the Envelope on Recommender Systems

Mohammad Khabbaz, Min Xie, Laks V.S. Lakshmanan
Department of Computer Science, University of British Columbia
{mkhabbaz,minxie,laks}@cs.ubc.ca

1 Introduction

Spurred by the advances in collaborative filtering, by applications that form the core business of companies such as Amazon and Netflix, and indeed by incentives such as the famous Netflix Prize, research on recommender systems has become quite mature and sophisticated algorithms that enjoy high prediction accuracy have been developed [1]. Most of this research has been concerned with what we regard as *first generation recommender systems*. Ever since the database community got interested in recommender systems, people have begun asking questions related to functionality. This includes developing flexible recommender systems which can efficiently compute top-k items within their framework [18] and using recommender systems to design packages subject to user specified constraints [11].

Significant effort has been dedicated to improving accuracy of recommendations. Many of the recommendation algorithms, while highly accurate, have scalability issues. The number of items managed by modern information systems is growing rapidly. Therefore, scalability is one of the serious issues for future generation recommender systems. Recommendation methods try to capture personalized patterns in user feedback data by making assumptions and keeping dense summaries of data. User feedback is typically represented in the form of a sparse matrix that stores existing ratings of users on items. There are two groups of methods – model-based and memory-based [1]. Model-based methods assume there is a lower dimensional underlying parametric model that has generated the ratings matrix. These methods aim at finding optimal parameter values for the model, given the observed data. Memory-based methods, on the other hand, calculate similarities between users or items and use these similarities for aggregating existing ratings and predicting unknown ratings. Thus, any item recommendation process has two steps: (1) an off-line training phase that captures personalized profiles (either as a model or as a similarity matrix); (2) an online recommendation generation process that uses the latest up-to-date model or similarity matrix to return top-k recommendations for a user. Any approach for improving scalability of item recommendation needs to pay attention to both profile building and recommendation generation. In section 2, we show how better scalability can be achieved in both aspects for one of the most popular and practical recommendation algorithms.

In addition to efficiency and scalability, an important limitation of classical recommender systems is that they only provide recommendations consisting of single items, e.g., books or DVDs. It has been recognized several applications call for composite recommendations consisting of sets, lists or other collections. For example, in trip planning, a user is interested in suggestions for a set of places to visit, or points of interest (POI). If the recommender system only provides a ranked list of POIs, the user has to manually figure out the most suitable set of POIs, which is often non-trivial as there may be a cost to visiting each place (time, price, etc.), and the

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

user may want the overall cost of all POIs to be less than a cost budget. Furthermore, some additional package constraints such as "no more than 3 museums in a package", "not more than two parks", "the total distance covered in visiting all POIs in a package should be $\leq$ 10 km." might render the manual configuration process even more tedious and time consuming. Another application which needs package recommendation is music list generation [17], where the system needs to recommend users with lists of songs called playlists, and users may have a constraint on the overall time for listening to these songs, and possibly constraints on the diversity of songs in the list.

So in these applications, there is a natural need for top-k package recommendations which can recommend users with high quality packages which satisfy all the constraints. Some so-called "third generation" travel planning web sites, such as NileGuide[1], YourTour[2], and some recent research works like [2, 3] are starting to consider certain of these features, although in a limited form.

1.1 Our Vision for the Next Generation Recommender System

We believe that the next generation recommender system should be efficient, scalable, and flexible enough to be tailed to different applications and users' customization requests such as the ability to compose packages and other collections and enforce user-specified constraints. In Figure 1, we show the architecture of our envisioned next generation recommender system that we call TopRecs$^+$.

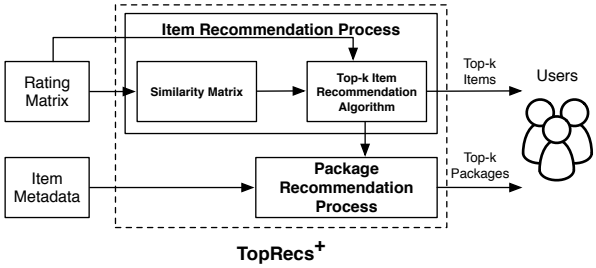

Figure 1: Next Generation Recommender System

In TopRecs$^+$, the recommendation engine can choose to recommend either top items or top packages depending on the application and user requests. The item recommendation engine can leverage the user item rating matrix to efficiently maintain the item-item similarity matrix, then based on this similarity matrix, an efficient and scalable top-k item recommendation algorithm can provide users with the set of k most interesting items [18]. On the other hand, based on the items generated by the item recommendation engine, combined with metadata information (such as price, type and etc.) associated with each item, the top-k package recommendation engine can return top-k packages that users will be interested in [11].

2 Top-k Item Recommendation

Predicting personalized scores of individual items for users is the core task of most recommendation algorithms. We follow item-based collaborative filtering (CF) [4], which is used widely in academia and practice [14, 5]. In CF, input data is typically represented as a sparse $n \times m$ matrix(R) of user ratings on items. An entry r_{ij} shows the existing rating of user u_i on item v_j. The main task is to predict the unknown ratings using the existing ones. Item-based CF computes and maintains an item-item similarity matrix using the existing ratings in R. Pearson correlation coefficient [12], is one of the popular choices for calculating item similarities. In item-based CF, the unknown rating $\widehat{r}_{ij}$, of u_i on v_j, is predicted by taking the weighted average of ratings of u_i on N most similar items to v_j. Equation 2 shows how existing ratings are aggregated to calculate $\widehat{r}_{ij}$ where $N(v_j, u_i)$ denotes the set of N items that are most similar to v_j and are rated by u_i.

[1] http://www.nileguide.com

[2] http://www.yourtour.com

$$\widehat{r}_{ij} = (\sum_{x=1}^{N} s_{xj} \times r_{ix}) / (\sum_{v_y \in N(v_j,u_i)} s_{yj}) \quad (2)$$

A unique challenge here in providing a score sorted list of items, is the fact that the individual scores to be aggregated to calculate $\widehat{r}_{ij}$ *come from different lists for different items.* This is because each candidate item can have a different set of N nearest neighbors among those rated by u_i. This makes it challenging to use traditional top-k algorithms. In fact, in [18], we theoretically show that adapting classic TA/NRA [13] algorithms, which are known to be instance optimal, for aggregating partial scores and returning top items leads to unpredictable performance due to this challenge. In particular, instance optimality no longer holds and there are instances where the adaptations can perform as bad as naive algorithms. Therefore, we identify the problem of discovering $N(v_j,u_i)$ for all candidate items to be the costly step in score prediction. For this purpose, we propose a novel algorithm, called the *Two Phase Algorithm (TPH)*. for finding all $N(v_j,u_i)$ in two steps.

2.1 Two Phase Algorithm

A naive approach for finding N nearest neighbors of a candidate item v_j in u_i's profile is to retrieve similarities of v_j to all (μ_i) items rated by u_i. Going over all μ_i similarity values and finding the N highest ones can be done in $O(\mu_i logN)$ for one item. t The total cost of finding nearest neighbors of all candidate items in u_i's profile is $O(m\mu_i logN)$, which can be costly in practice if μ_i is large.

In order to design a more efficient process, we propose a new global data structure, L, rather than the similarity matrix. Every column of L corresponds to one of the items. Items in each column are sorted using their similarities with respect to the item indexing the column. Thus, the j^{th} entry in the i^{th} column of L is a pair (item-id, sim), where item-id is the id of the j^{th} most similar item to the i^{th} item. The second element of the pair is the similarity between these two items.

The main intuition behind the two phase algorithm is the following. Assuming that some $N' < \mu_i$ nearest neighbors of a candidate item v_j are known, finding N nearest neighbors can be done more efficiently. This is regardless of whether N' is greater or smaller than N. Suppose we know $N' < N$ nearest neighbors of v_j, then finding the remaining ones can be done in $O(\mu_i log(N-N'))$. If $N' = N$, no further processing is needed. For $N' > N$, it again takes $O(N' logN)$ to find the N nearest neighbors.

All of the required similarity values for finding N nearest neighbors are in the μ_i columns of L that correspond to rated items. Therefore, we propose our two phase algorithm as follows. In the first phase, we choose a similarity threshold and read only those values from these columns that are above the threshold. This leads to discovering a variable number of nearest neighbors for every candidate item. Depending on the number of neighbors found for each item, in the second phase we find the exact N nearest neighbors. For those items that we have managed to find some neighbors for, the process will be more efficient as described earlier.

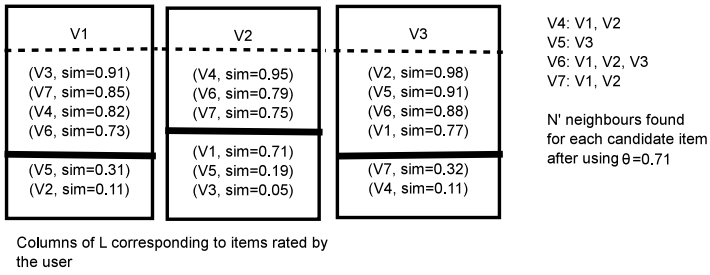

Figure 2: An example of running the two phase probe step using a prob threshold of $\theta = 0.72$ and comparing it to naive algorithms

Figure 2 illustrates the process using a threshold value $\theta = 0.72$. In the first phase all of the entries above

the threshold are read as shown on the left side. In this example $\mu_i = 3$. Notice that for both cases of $N = 1$ or 2, the process can be done more efficiently for three out of four candidate items.

2.2 Optimal Threshold θ

The cost of the two phase algorithm ($C(\theta)$) can be written as the sum of three main components: (1) Cost of the first phase ($C1(\theta)$); (2) Cost of finding N nearest neighbors when $N' < N$ ($C2(\theta)$); (3) Cost of finding N nearest neighbors when $N' > N$ ($C3(\theta)$).

For instance, assuming the example in Figure 2 and $N = 2$, v_5 falls in the second category and v_6 falls in the third category. While for v_4 and v_7, we have already found their 2 nearest neighbors. Using smaller θ results in greater $C1$ and $C3$ and smaller $C2$. This is due to the fact that more similarity values will pass the filter and make it to the second phase. On the other hand, $C2$ increases if we use a larger threshold and the other two components decrease. Therefore, there is a tradeoff between $C1$ and $C3$ on one hand and $C2$ on the other. Optimal threshold value is one that minimizes the total cost of all components put together.

We perform a probabilistic cost-based analysis in [18], in order to find the optimal threshold value. In [18], we fit a Gaussian probability distribution to the similarity values in the similarity matrix. Using the cumulative density function, we calculate the probability that a particular similarity value can result in one of the N nearest neighbors of another item. Our cost function provides an upper bound on the expected cost of both phases together. We find the optimal similarity threshold that minimizes $C(\theta)$. Moreover, we theoretically prove that due to the tradeoff between cost components, $C(\theta)$ always has one and only one minimum. We refer the reader to [18] for more details, where we also empirically evaluate our algorithm. Our empirical results confirm the reliability of our theoretical probabilistic process for finding the optimal threshold value which in turn leads to a consistently efficient performance of the top-k recommender algorithm.

2.3 Updating the Similarity Matrix

Pearson correlation, Cosine and Jaccard are some of the popular examples of similarity measures used in CF. Among all of these, Pearson correlation has been applied most widely in practice. It is possible to provide guidelines in order to keep the similarity matrix updated for most of these measures. Here, we show how this is doable for Pearson correlation measure. Equation 3 shows how similarity between two items v_i and v_j is calculated using Pearson correlation coefficient. It measures the similarity between two items using only ratings of users who have rated both items (I_{ij}).

$$s(i,j) = \frac{\sum_{u \in I_{ij}} (R(u,v_i) - \bar{r}_{v_i})(R(u,v_j) - \bar{r}_{v_j})}{\sqrt{\sum_{u \in I_{ij}} (R(u,v_i) - \bar{r}_{v_i})^2 \sum_{u \in I_{ij}} (R(u,v_j) - \bar{r}_{v_j})^2}}, I_{ij} = v_i \cap v_i \tag{3}$$

Equation 4 provides the sufficient statistics that can be stored in order to be able to update the similarity matrix incrementally.

$$A_{ij} = \sum_{u \in I_{ij}} R(u,v_i)R(u,v_j), B_{ij} = \sum_{u \in I_{ij}} R(u,v_i), C_i = \bar{r}_{v_i}, D_{ij} = \sum_{u \in I_{ij}} R(u,v_i)^2, E_{ij} = |I_{ij}| \tag{4}$$

Keeping the similarity matrix updated requires storing four $m \times m$ matrices (A, B, D, E) and a vector of size m (C), containing averages of values as shown in Equation 4. When a new rating becomes available, all of the matrices in Equation 4 can be updated incrementally. Equation 5 shows how every entry of the similarity matrix can be written based on the matrices defined in Equation 4.

$$s(i,j) = \frac{A_{ij} - C_j B_{ij} - C_i B_{ji} + E_{ij} C_i C_j}{\sqrt{(D_{ij} + E_{ij} {C_i}^2 - 2C_i B_{ij})(D_{ji} + E_{ij} {C_j}^2 - 2C_j B_{ji})}} \tag{5}$$

The similarity matrix can be updated incrementally with respect to a new rating in two steps: (1) all of the entries of matrices in Equation 4 that are affected by the new rating are updated; (2) all of the entries of the similarity matrix that are affected by the changes in step 1 are updated. Since we are keeping similarities in a data structure L and columns of L are kept sorted by similarity, every column of L could be stored as a priority queue. Providing efficient strategies for performing these updates in terms of computational cost, quality of results and memory requirements is part of our ongoing work.

3 Top-k Package Recommendation

As mentioned in the introduction, many applications like trip planning and music list generation can benefit from having packages recommended instead of a ranked list of single items. In this section, we will discuss how the package recommendation engine can be built upon the item recommender system.

Let $\mathcal{I}$ be the set of all items, for each item $v \in \mathcal{I}$, we denote the *value* of v for the current active user as $val(v)$ which can be obtained as the predicted utility or rating from the underlying item recommendation algorithms. We denote the *cost* of v as $c(v)$. The cost may correspond to time, price, etc. For a subset of items $P \subseteq \mathcal{I}$, we define the *value* of P as $val(P) = \sum_{v \in P} val(v)$, and the *cost* of P as $c(P) = \sum_{v \in P} c(v)$. Let $\mathbf{P} = \{P \mid P \subseteq \mathcal{I}\}$ be the set of all possible subsets of items, and given a user defined cost budget B, a subset of items $P \subseteq \mathcal{I}$ is *feasible* if $c(P) \leq B$. We can define our top-k package recommendation problem as follows.

Definition 1: (Top-k Package Recommendation): Given a universe of items $\mathcal{I}$ and an underlying item recommender system for predicting values of items for the current active user, a cost budget B, find top-k packages $P_1, ..., P_k$ such that each P_i is feasible and $val(P) \leq val(P_i)$ for all feasible packages $P \in \mathbf{P} - \{P_1, ..., P_k\}$.

As discussed in [11], when $k = 1$, the top-k package recommendation problem can be viewed as a variation of the 0/1 knapsack problem [6], where we have the restriction that items can be accessed only in the non-increasing order of their value. This is because of the way recommendations are made by the underlying item recommender system. Furthermore, because solving the top-k package recommendation problem optimally is NP-hard [6], we need to consider approximate answers instead of exact ones, i.e., in Definition 1, for a package P_i in the top-k package set, instead of requiring $val(P) \leq val(P_i)$ for all feasible packages $P \in \mathbf{P} - \{P_1, ..., P_k\}$, we aim for $val(P) \leq \alpha \times val(P_i)$, where α is the approximation factor.

Let c_s be the access cost of retrieving the next highest-value item from the underlying item recommender system and let c_r be the access cost of obtaining the cost (time, price etc) associated with an item. It is clear that total access cost of processing n items is $n \times (c_s + c_r)$. Notice that c_s and c_r can be large compared to the cost of in-memory operations: for both accesses information may need to be transmitted through the Internet, and for the sorted access, $val(v)$ may need to be computed. So well-known algorithms for knapsack which need to access *all* items [6] may not be realistic, and instead we should minimize the total number of items accessed by the algorithm and yet ensure that high quality top-k packages are obtained.

In [11] we also show that without background knowledge about the cost distribution of items, in the worst case, we must access all items to find top-k packages. To facilitate the pruning of item accesses, we thus assume some simple background information $\mathcal{BG}$ about item costs. In [11], we assume $\mathcal{BG}$ is the minimum item cost for illustrative purposes, however, more sophisticated stats like histogram can be easily incorporated.

In the rest of this section, we will first consider an algorithm which can minimize the number of items accessed while guaranteeing the quality of the packages returned, then we will discuss a greedy algorithm which may not be optimal w.r.t. the number of items accessed but has very good empirical performance. Handling of additional constraints and highlights of the empirical results will be discussed at the end of this section.

3.1 Instance Optimal Algorithm

As discussed in the previous section, it is crucial for an algorithm to find high-quality solutions *while minimizing the number of items accessed*.

Consider the top-1 package recommendation problem. Let $S = \{v_1, ..., v_n\}$ be the set of items which have been accessed so far. It is well known from previous work [6] that a pseudo-polynomial algorithm can be utilized to get the optimal knapsack solution for S. Furthermore, as shown in [11], by utilizing the same algorithm and the background information $\mathcal{BG}$, we can also get a tight upper bound V^* on the value of the optimal solution. So we can have an iterative algorithm which retrieves a new item in each iteration, calculate the optimal solution R^o for S and stop the algorithm once $val(R^o) \geq \frac{1}{\alpha} \times V^*$.

We have shown in [11] that the above algorithm can correctly return approximate top-k package recommendations, and is *instance optimal* [13]. In particular, given any instance of the problem, and given any α-approximation algorithm A for the problem with the same background cost information $\mathcal{BG}$ and the same access constraints, A must read at least as many items as the our algorithm.

To produce top-k package recommendations, we can apply Lawler's procedure [19] to the above top-1 package recommendation algorithm. The idea is that instead of returning the α-approximation solution found in the top-1 package recommendation algorithm, we enumerate at this point all possible α-approximation solutions using Lawler's procedure. If the number of α-approximation solutions is at least k, then we can report the top-k packages found; otherwise, we continue accessing the next item. It can be shown that the above top-k package recommendation algorithm is also instance optimal.

3.2 Greedy Algorithm

Although the instance optimal algorithms presented above guarantee to return top-k packages that are α-approximations of the optimal packages, they rely on an exact algorithm for the knapsack problem, which may lead to high computational cost. To remedy this, we proposed in [11] more efficient algorithms which utilize simple greedy heuristics to form a high quality package from the accessed itemset S.

Similar to the instance optimal algorithm, this greedy algorithm will always generate a correct α-approximation to the optimal solution, however, it is not instance optimal among all α-approximation algorithms with the same constraints and background information.

3.3 Highlights of the Empirical Results

To evaluate the performance of various proposed algorithms, in [11] we tested our algorithms on four datasets: two real datasets MovieLens[3], TripAdvisor[4]; and two synthetic datasets, one with item value correlated with item cost, and another with item value and item cost uncorrelated.

We first tested the quality of packages generated by our proposed algorithms compared with optimal top-k packages generated using an offline knapsack solver. It can be verified that our proposed approximation algorithms can return top-k packages whose values are very close to the optimal solution. Furthermore, by comparing the average item value from the top-k packages generated, it can be verified that our proposed approximation algorithms often recommend packages with high average value, whereas the optimal algorithm often tries to fill the package with small cost and small value items.

For MovieLens, TripAdvisor and the uncorrelated dataset, we have verified that on average the greedy algorithm has excellent performance in terms of *both* running time and access cost. The instance optimal algorithm also has low access cost, but its running time grows very quickly with k since it needs to solve exactly many instances of knapsack, albeit restricted to the accessed items. The only dataset where both the greedy and instance optimal algorithms have a high access cost is the correlated dataset (but for this case the greedy algorithm still has good running time). The reason for this is that, for the correlated dataset, the global minimum cost corresponds only to items which also have the least value. Thus the information it provides on the unseen items is very coarse. In practice, an obvious solution to this is to obtain more precise background cost information.

[3] http://www.movielens.org

[4] http://www.tripadvisor.com

3.4 Handling Additional Package Constraints

For many applications of package recommendation, the users might have some additional constraints, e.g., for trip planning, the user may require the returned package to contain no more than 3 museums. To capture these constraints in our algorithms, we can define a Boolean *compatibility function* $\mathcal{C}$ over the packages under consideration. Given a package P, $\mathcal{C}(P) = true$ iff all constraints on items in P are satisfied. We can add a call to $\mathcal{C}$ in the proposed algorithms after each candidate package has been found. If the package fails the compatibility check, we just discard it and search for the next candidate package.

It is worth noting that many constraints studied in the previous work such as [2] and [9] are restricted classes of boolean compatibility constraints. However, depending on the application needs, for scenarios where only one specific type of constraint is considered, e.g., having one item from each of 3 predefined categories, more efficient algorithms like Rank Join [16] can be leveraged.

4 Related Work

Item-based CF was first proposed in [4]. Their solution for improving scalability is storing only N (between 10 and 30) nearest neighbors of each item regardless of user profiles. This approach improves scalability for some items but has a major drawback: for some users we may not be able to find any rated items among these N items and make predictions. Other works have tried to improve scalability by instance selection [7, 8]. The main idea is finding nearest neighbors of all items from a smaller subset of items, thus the nearest neighbors are not global nearest neighbors. It is also worth mentioning that our approach achieves scalability without deviating from the standard item-based method, while instance selection achieves scalability at the cost of sacrificing accuracy. Our approach is orthogonal to instance selection. Indeed, both methods can be combined for providing better scalability when accuracy can be traded for space needed for storing all pairwise item similarities.

For package recommendation, the closest to our work is [2], where they are interested in finding top-k tuples of entities. Examples of entities include cities, hotels and airlines, while packages are tuples of entities. A package in their framework is of fixed size, e.g., one city, one hotel and one airline, with fixed associations among the entities essentially indicating all possible valid packages. Instead, we allow for packages (composite recommendations) of variable size, subject to a budget constraint. Associations between entities can be easily captured in our framework using the notion of compatibility of sets. Another closely related work is [9] where a novel framework is proposed to automatically generate travel itineraries from online user-generated data like picture uploads. They formulate the problem of recommending travel itineraries of high quality where the travel time is under a given time budget. However, in this work, the value of each POI is determined by the number of times it is mentioned by users, whereas in our work, item value is a personalized score which comes from an underlying recommender system and unlike their work, accessing these items is constrained to be in value-sorted order. Finally, motivated by online shopping applications, [15] studies the problem of recommending "satellite items" related to a given "central item" subject to a cost budget. The resulting notion of packages is quite restricted compared to our framework, and item values are not taken into account.

5 Summary and Open Problems

In this article, we have discussed our vision for the next generation recommender system which has an efficient and scalable item recommendation engine which recommends top-k interesting items, and an efficient package recommendation engine which recommends top-k interesting packages which satisfy all the user specified constraints. While we have investigated some initial efforts in realizing such a recommender system [18, 11], much remains to be done for realizing our vision. In the following, we will point to a few interesting open problem in this direction.

- In order to handle dynamically changing user-item ratings matrix, an efficient way of maintaining the similarity matrix is critical. Preliminary ideas for doing this appear in [18]. Efficient algorithms for

incremental maintenance of the similarity matrix is an important open problem.

- Instead of returning the exact top-*k* answers, for many cases, we might also be interested in providing approximate top-k recommendations with probabilistic guarantees. Some previous work like [10] has investigated probabilistic threshold algorithm, however, it cannot be directly utilized in a recommendation framework. A related point is materializing and maintaining a "key" subset of similarity entries instead of the whole similarity matrix, thus trading accuracy for speed and storage.
- For top-*k* item recommendation algorithms based on model-based methods such as matrix factorization [1] is an interesting problem.
- For top-*k* package recommendations, in [11] we have investigated the problem of recommending sets of items. Sometimes, the order of items in the recommended package might be also critical, e.g., as mentioned in [17], the order of songs in the recommended music list might be important for the users; and also for trip planning, the order of visiting different POIs can be important. Thus, recommendation of richer types of collections should be investigated.

References

[1] G. Adomavicius and A. Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE TKDE*, 17(6):734–749, 2005.

[2] A. Angel et al. Ranking objects based on relationships and fixed associations. In *EDBT*, 2009.

[3] A. Brodsky et al. Card: A decision-guidance framework and application for recommending composite alternatives. In *ACM RecSys*, 2008.

[4] B. Sarvar et al. Item-based collaborative filtering recommendation algorithms. In *WWW*, 2001.

[5] C. Yu et al. It takes variety to make a world: Diversification in recommender systems. In *EDBT*, 2009.

[6] H. Kellerer et al. *Knapsack Problems*. Springer, 2004.

[7] K. Yu et al. Instance selection techniques for memory-based collaborative filtering. In *SDM*, 2002.

[8] K. Yu et al. Probabilistic memory-based collaborative filtering. *IEEE TKDE*, 16(1):56–69, 2004.

[9] M. De Choudhury et al. Automatic construction of travel itineraries using social breadcrumbs. In *ACM Hypertext*, 2010.

[10] M. Theobald et al. Top-k query evaluation with probabilistic guarantees. In *VLDB*, 2004.

[11] M. Xie et al. Breaking out of the box of recommendations: From items to packages. In *ACM RecSys*, 2010.

[12] P. Resnick et al. Grouplens: An open architecture for collaborative filtering of netnews. In *CSCW*, 1994.

[13] R. Fagin et al. Optimal aggregation algorithms for middleware. In *PODS*, 2001.

[14] S. Amer-Yahia et al. Group recommendation: Semantics and efficiency. In *PVLDB*, 2009.

[15] S. Basu Roy et al. Constructing and exploring composite items. In *SIGMOD*, 2010.

[16] J. Finger and N. Polyzotis. Robust and efficient algorithms for rank join evaluation. In *SIGMOD*, 2009.

[17] D. Hansen and J. Golbeck. Mixing it up recommending collections of items. In *CHI*, 2009.

[18] M. Khabbaz and L. Lakshmanan. Toprecs: Top-k algorithms for item-based collaborative filtering. In *EDBT*, 2011.

[19] E. Lawler. A procedure for computing the k best solutions to discrete optimization problems and its application to the shortest path problem. *Management Science*, 18(7):pp. 401–405, 1972.

Recommendation Projects at Yahoo!

Sihem Amer-Yahia,
Yahoo! Labs

1 Introduction

Recommendation is the task of providing content that is likely to interest users and enrich their online experience. At Yahoo!, the variety of user-facing applications and the nature and volume of data raise multiple recommendation opportunities and challenges. In this paper, we provide a brief description of several projects within Yahoo! Labs, in the context of Web Search (Section 2), interpreting users' clicks and browsing behavior (Section 3), and on the Social Web (Section 4).

2 In Web Search

2.1 Diversifying Presubmit Query Recommendations
Umut Ozertem, Emre Velipasasaoglu

Search assistance is an important feature in commercial search engines. Its two main flavors are *presubmit* and *postsubmit*. In presubmit, the aim is to auto-complete a user's partial query (referred to as prefix hereafter) to save time. In postsubmit, the goal is to suggest relevant follow-up queries to a user query. Presubmit poses a harder problem because (i) input from the user is a partially typed query and the user's intent is not defined, (ii) the algorithm needs to run at each character stroke, imposing a very strict computational complexity bound.

In Web search ranking, queries are associated with multiple intents, and when a user's intent is unknown, trading-off some relevance for result diversity is desirable [4, 6, 13, 32]. Presubmit is a clear case where intent is not defined. Common solutions suggest the most popular queries that match a user's prefix. However, mere frequency sorting leads to many cases with low utility (see Table 1). Therefore, in order to minimize effort (number of characters typed) before finding a useful suggestion, one needs to introduce diversity into the suggestion set.

The need for diversification has been studied in the context of query suggestion. There are two recent publications on this topic: one based on random walks over the query-URL bipartite graph [30], the other based on manifold ranking [38]. These methods define the diversification objective as "*given a query, generate a relevant but also diverse suggestion set*". Neither method is suitable for presubmit, where the query is yet to be defined. Hence, we adapt the problem to our context and define it as "*given a set of suggestions of size N, rerank the suggestions such that the overall utility in top K rank is maximized for all* $K \leq N$."

Diversity and relevance are conflicting objectives and optimizing both of them is known to be NP-hard. We develop a greedy algorithm that picks the query with highest marginal utility at each step [31]. We define a query utility as a function of the difference between its Web results and those of already presented query suggestions. For example the queries "facebook login" and "facebook home" are recognized as low utility queries when the

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

Table 1: Suggestions for prefixes **faceb** and **well** sorted by frequency and diversification

Frequency for **faceb**	Diversification for **faceb**		Frequency for **well**	Diversification for **well**
facebook	**faceb**ook		**well**s fargo	**well**s fargo
facebook login	farmville **faceb**ook		**well**s fargo bank	**well**sbutrin
facebook home	**faceb**ook farm town		**well**s fargo online	tom **well**ing
faceboklogin	**faceb**ook layouts		**well**sbutrin	**well**point
facebook friends	**faceb**ook ipo		tom **well**ing	dawn **well**s

query "facebook" precedes them. We define a measure of utility based on a user browsing model, and estimate the probability that each result can be examined in the search result page of the other query.

To build our model, we used 6 months of anonymized session logs of Yahoo! Search. We ignored queries that appeared less than 3 times. The model takes about 2.7M unique queries and 55.6M unique URLs in total, and builds a 2.7M $\times$ 2.7M query similarity matrix. We threshold this similarity matrix to construct a look-up table of duplicate queries. The look-up table consists of 1.56M queries with 20 duplicates each on average.

We designed an A/B test to measure effectiveness. The search engine traffic is split into two groups: the baseline (frequency sorting) and the control (greedy diversification algorithm) buckets. The user populations in both groups are uniformly sampled and their query volumes and distributions, i.e., the proportion of navigational and informational queries, are similar. If the greedy diversification algorithm changes the query suggestions we record this event as an *affected query*. Over affected queries, we found that the average character savings increased by 17%[1], and the time it takes the user to click on a destination URL decreases one second on average. A one-second time saving per search is considerable given the overall search volume. Over all the control bucket traffic, the increase in latency due to the diversification algorithm is only 1%.

2.2 Entity Ranking for Web Search

Changsung Kang, Srinivas Vadrevu, Ruiqiang Zhang, Roelof van Zwol, Lluis Garcia Pueyo, Nicolas Torzec, Jianzhang He, Yi Chang

A recent trend in Web search is to return structured entities. For example, a user looking for a movie can also see the movie plot, genre, cast, review and show times at her location while displaying related news articles, images, videos and tweets, wherever possible. The type of shown entities depends on the category of the query entity. For example, for movie queries, the goal is to show both lateral information like related movies and faceted information such as its cast.

In this work, the extraction of relevant entities relies on an Entity-Relationship knowledge base. The knowledge base contains an Entity-Relationship graph where nodes are entities and their attributes, and edges are semantic relationships among them. Our base consists of 4 million entities (100+ fine-grain entity types organized in a taxonomy) and 80 million relationship instances (hundreds of fine-grain relationship types, including both first order and derived relations). The main domains covered include Yahoo! Movies, Yahoo! TV, Yahoo! Music, Yahoo! Sports, Yahoo! OMG, and Yahoo! Travel. We have in the order of several hundreds facets for popular entities, and a few dozens for infamous ones.

We build a framework that computes entity features from various sources, including search logs, Flickr tags, and entity pairs detected in Twitter. From each source, we compute several probabilistic features that indicate the strength of the relationship between two entities based on their co-occurrence in various sources. Examples of features include atomic ones such as entity probability or entropy, symmetric features such as point-wise mutual information, asymmetric features such as conditional probability, and compound features that combine multiple ones. Two additional features are also computed from the graph: entity popularity based on a PageRank-like approach, and shared connections that indicate relationship strength in the graph.

[1] Average query length is 22, average prefix length for a suggestion click decreases from 13.5 to 12 after diversification.

We cast entity ranking as a supervised machine learning problem [37] with the goal of predicting relevance of the related entity to the query entity. We incorporate the categories of related entities into the loss function and leverage related entities from different categories to improve relevance.

Two types of experiments were conducted to validate our algorithms. The first set evaluates the performance of our approach on editorially judged entity pairs and the second aggregates user behavior on search results. Our data set contained 6000 query entities and 33000 entity-facet pairs. In the first experiment, editors provided a five-point relevance grade to indicate the match between a query entity and a facet entity. We showed that we achieve high Discounted Cumulative Gain (DCG). In the second experiment, we used pair-wise accuracy that reports how often users prefer a higher ranked entity over lower ranked entities in the related entity results. Comparison with several click preference models such as Cumulative Relevance [18], Skip CTR [22] showed the superiority of our Pair-wise Comparison Model. More details on this work can be found here [23].

3 Interpreting User Clicks and Browsing Behavior

3.1 Ad Recommendation using the Query-Ad Click Graph
Tasos Anastasakos

We address the sponsored search retrieval problem that is, the problem of finding and ranking relevant advertisements to a search query. A text ad on a search results page, otherwise known as sponsored link, is treated as a short document with a title, a short text and some keywords. We describe a technique to determine the relevance of an ad document to a search query using click-through data. The method builds on a collaborative filtering approach to discover and recommend new ads related to a query using a click graph. It is implemented on a graph with several million edges and easily scales to larger sizes easily.

Major search engines typically see traffic that contains millions of queries and a correspondingly large number of user-clicks on documents shown as results to those queries. While clicks are noisy due to click fraud, accidental or exploratory clicks, and position-bias, they can often be interpreted as a weak indicator of relevance. Several recent works have used click information to improve performance on various tasks. We propose an algorithm that operates on a large bipartite graph of queries and documents, where an edge between a query and a document is determined by whether users have clicked on the given document for the query. The weight on the edge represents the strength of the association measured as a function of several user metrics such as the number of clicks generated for the corresponding query and ad and the click-through-rate (CTR) of the query ad pair. We apply a collaborative filtering approach to first determine query-query similarity on this bipartite graph. Subsequently these query similarity scores are used to discover new documents that have not been shown for a given query before [7, 8].

The click-graph is built from a portion of Yahoo!'s sponsored search traffic over a 2-week period. It contains user queries and associated displayed and clicked ads. A typical graph consists of 27 million unique queries, 20 million unique ads and 51 million query-ad edges. The graph is regenerated every week in order to include recent queries and recent changes in user activity. We implemented a collaborative filtering algorithm on Map-Reduce and output a large lookup table of query-query and query-ad associations. (provide more brief details on algorithm or it will be hard to understand why you are doing better than baselines)

We evaluated the ad recommendations from our work off-line using human annotators/editors and online using live Web traffic to measure CTR improvements. We summarize our evaluation below.

Editorial Assessment : Query suggestions were labeled by trained editors who gave each ad a rating from 1 to 7, to reflect how strongly the ad meets the likely commercial intent or explicit need of the query. A rating of 1 is reserved for cases where only one perfect ad matches the query (e.g.,). Ratings 2 to 7 represent a decreasing degree of relevance.

Online Testing We ran our system live on a fraction (or "bucket") of Yahoo!'s traffic. The online traffic was divided so that a random subset of the user population (or a "bucket") receives candidates retrieved by the new algorithm. Our evaluation involved several millions users and collected statistics over a sufficiently long period of time (specify the time period). The goal of the experiment was to evaluate the monetization capability of the new algorithm using bucket testing measures [26].

We compared our algorithm to several baselines. We found 2-18% relative improvements in click rates, and 5-37% relative improvements in editorial judgments compared to the baseline.

Directions for future work include incorporating negative feedback and improving the performance for tail queries.

3.2 User Action Interpretation for Personalized Content Optimization

Anlei Dong, Jiang Bian, Srihari Reddy

Content optimization for recommender system is known as the problem of selecting content items to present to a user who is intent on browsing for information. Examples of content optimization are article publishing on portal Websites [2, 1], news personalization [16, 27], recommendation of dynamically changing items (updates, tweets, etc), and computational advertising [11, 33]. This work will address the variant that displays the best set of trending queries from the search engine in a module on the portal Website. This application is different from the task of query suggestion in Web search in the sense that it recommends popular queries to users from a certain pool of globally trending queries while query suggestion suggests queries relevant to those just submitted.

We are interested in user engagement when visiting a page. Engagement means the user examined or at least partly examined the content of a visited Web page. For example, when a user visits Yahoo! front page, it is possible she totally ignores the displayed *Trending Now* module content as she may be attracted to other modules such as the *Today* module, or goes to other services such as *Search* and *E-mail*. For a recommendation module, accurate Click-Through-Rate (CTR) estimation should be based on the events where users were really engaged, instead of all the events where the content of this module were displayed to users. We identify two categories of events regarding user engagement: *click event* and *non-click event*. In click events, it is obvious that the user likes the item she clicked, it also implies that other non-clicked items inside this module are not so interesting to the user compared to the clicked item. In contrast, non-click events are less informative because the user might not have examined the related module at all and the system cannot infer whether or not she is interested in the items belonging to the module. In this work, we only use click events as user feedback for online CTR estimation.

To validate our argument, we conduct experiments on real data from *Trending Now* on Yahoo! homepage. All data is anonymized in accordance with Yahoo!'s policy. We collected events in terms of *views* and *clicks* from a random learning bucket ([15]) during ten days from November 30th, 2010 to December 9th, 2010. This resulted in hundreds of millions of events with many millions unique users. Queries are randomly and displayed to users. An event records a user's action on the served queries on *Trending Now* ("*view*" is encoded as -1, "*click*" as 1). Specifically, we represent each event e as a set of tuples:

$$e = \langle u, t, p, q_p, a \rangle,\ p = 1, 2, \ldots, 10$$

where u denotes the user, t represents the time stamp for this event, q is the served query, p is the position at which q is displayed (there are totally ten positions on *Trending Now*), a is either *view* or *click*.

Users are assigned to groups according to a segmentation model. In this model, the system serves users in a group with models updated using feedback from users belonging to the group. User grouping is determined based on their demographics (e.g., age, gender) and other behavior and preference features [15]. To evaluate the recommendation model for *Trending Now*, we use the model to predict ranking scores for all candidate queries at a certain time stamp of each event, and rank them in descending order. For click events, we measure the

ranking position of the query that has actually been clicked by the user. More formally, for those clicks that actually happened at Position 1, we define $precision_i$ as the number of these clicks whose corresponding clicked items are ranked at Position up to i by the prediction of the model. This evaluation metrics has been proved to be unbiased towards online result [28].

We compare the modeling results using both all events and click events, where the model using all events is the baseline model. We observe from Table 2 that only using click events can significantly improve CTR estimation.

Table 2: Relative precision gain when training using click events over training on all events

Model	$prec_1$	$prec_2$	$prec_3$	$prec_4$	$prec_{10}$
Use click events	11.11%	7.05%	8.22%	7.70%	6.67%

Much research on user action interpretation has been conducted in the context of Web search. In particular, online user behavior modeling has been attracting much attention in recent years. Some work uncovered user behavior models based on controlled user studies [21, 35], while other studies focused on large-scale log analysis [36, 17]. Recently, some research [20, 19] has used eye-tracking studies to understand in detail how searchers examine search results, meanwhile dwell time interpretation has also attracted significant attention [25, 24] and has been extensively used for various information retrieval tasks [12, 29, 3]. However, user action interpretation has not been paid much attention in the studies of content optimization. Our work proposes to take deep analysis on user action interpretation in recommender system. In particular, we leverage user behavior information to sample training examples in order to remove those with little benefit for learning the model. To our best knowledge, none previous work has studied interpreting user actions in the context of content optimization.

4 On the Social Web

4.1 Recommending Travel Itineraries from Flickr Photos

Sihem Amer-Yahia, Nadav Golbandi, Ronny Lempel

Vacation planning is one of the frequent—but nonetheless laborious—tasks that people engage themselves with online; requiring skilled interaction with a multitude of resources. This paper constructs intra-city travel itineraries automatically by tapping a latent source reflecting geo-temporal breadcrumbs left by millions of tourists. For example, the popular rich media sharing site, Flickr, allows photos to be stamped by the time of when they were taken and be mapped to Points Of Interests (POIs) by geographical (i.e. latitude-longitude) and semantic (e.g., tags) metadata.

Leveraging this information, we construct itineraries following a two-step approach. Given a city, we first extract photo streams of individual users. Each photo stream provides estimates on where the user was, how long he stayed at each place, and what was the transit time between places. In the second step, we aggregate all user photo streams into a POI graph. Itineraries are then automatically constructed from the graph based on the popularity of the POIs and subject to the user's time and destination constraints [14].

We evaluated our approach by constructing itineraries for several major cities and comparing them, through a "crowd-sourcing" marketplace (Amazon Mechanical Turk), against itineraries constructed from popular bus tours that are professionally generated. Our extensive survey-based user studies over about 450 workers on AMT indicate that high quality itineraries can be automatically constructed from Flickr data.

This work is being integrated into Yahoo! Travel for suggesting intra-city itineraries of different time lengths. In particular, it is leveraged on mobile devices forinteractive itinerary planning as in [34].

4.2 Centralized and Distributed Query Personalization on Delicious
Sihem Amer-Yahia, Xiao Bai

Collaborative tagging systems represent huge mines of information. Yet, exploring these mines is challenging because of the unstructured nature of tagging and the lack of any fixed ontology. An appealing way to reduce the exploration space in collaborative tagging systems is to personalize the search by exploiting information from the social acquaintances of the seeker, typically users that exhibit similar tagging behaviors.

Given a user and a so-called user's network, composed of her social acquaintances, the relevance of an item to the user's query is a function of its popularity in that network, weighted by social distance. This challenge has been explored in both centralized and peer-to-peer environments. In [5], it was shown that scalable and efficient network-aware search must leverage user behavior to balance storage volume and response time. While search over information popular in a seeker's network can be achieved efficiently with straightforward adaptations of well-known algorithms, the storage volume of indexing every (seeker, keyword) pair, is daunting. Two space-saving solutions were explored: network clustering and behavior clustering.

In a centralized environment, it is expensive to maintain the inverted lists up-to-date in highly dynamic collaborative tagging systems where users frequently change their profiles by tagging new items. One possibility to circumvent this problem and improve the scalability is to personalize the search in a decentralized way. Besides being scalable and able to cope with dynamics, decentralized solutions inherently circumvent the danger of central authorities abusing the information at their disposal.

A natural, fully decentralized solution would consist for each user in a peer-to-peer system to locally store and maintain her network, enabling thereby efficient top-k query processing. The experiment on a 10,000-user *delicious* trace [9] confirms that the same result quality and query response time, as the most time efficient but storage consuming centralized approach, can be achieved with 0.011% of the total data stored at each user. Storage is no longer a severe issue in a decentralized setting. Yet, this requires each user to store all the profiles of her social acquaintances: these would then be massively replicated and hence hard to maintain. At the other extreme, a storage-effective strategy would consist for each user to store and maintain only her own profile and seek other profiles whenever a query is to be processed. Clearly, this optimizes the storage and maintenance issues but might induce a large number of messages and a large latency if profiles of acquaintances are to be consulted at query time. In addition, the profiles of temporarily disconnected users would be unavailable which, in turn, might significantly hamper the accuracy of the query processing.

A pragmatic solution [10], which we call P3Q (Peer-to-Peer Personalized Query processing), is a bimodal, gossip-based protocol to personalize the query processing in peer-to-peer systems. Users in P3Q periodically maintain their networks of social acquaintances by gossiping among each other and computing the proximity between tagging profiles. Each user maintains her network, namely a set of IDs of her social acquaintances. A user however only locally stores a limited subset of profiles, according to her storage capability. This removes on the one hand the bottleneck of a central server when the system keeps growing, and avoids on the other hand burdening any individual user in terms of storage. The maintenance of the network is performed in a lazy gossip mode, at a fairly low frequency to avoid overloading the network.

The querying scheme itself is based on an eager mode of the gossip protocol, i.e., with an increased frequency, and is biased towards social acquaintances. Each query is first computed locally by the querier, based on the set of stored profiles, providing an immediate partial result. Then the query, together with the list of profiles needed to compute it, is gossiped and computed collaboratively. The results are iteratively refined accordingly. Gossiping the query avoids saturating the network by contacting all the users in the personal network at the same time, and refreshes the part of the network originating from the querier, generating a specific wave of refreshments in the personalization process.

The analysis shows that the storage at each user only depends on her storage capability and does not increase with the system scale. The query processing time in gossip cycles can be approximated with $O(log_2 L)$, where L is the number of profiles in a user's network that contribute to the query processing but are not stored by

her. The experimental evaluation in a 10,000-user *delicious* trace confirms the scalability of P3Q: even if each user stores only 10 profiles out of thousands of social acquaintances in her network, corresponding to 12.5M bytes without any compression technique, the top-k queries can be accurately satisfied within 10 gossip cycles. Running the lazy mode every minute, even if all users simultaneously change their profiles, in half an hour, 95% of the stored profiles are updated, ensuring the freshness of the query results. Meanwhile, P3Q incurs acceptable overload in terms of bandwidth consumption: 13.4 Kbps are sufficient for maintaining the network and 91 Kbps are sufficient to compute a query. More adequate bandwidth, allowing both modes to run in higher frequency, would significantly decrease the network maintenance time as well as the query processing time.

References

[1] D. Agarwal, B.-C. Chen, and P. Elango. Fast online learning through offline initialization for time-sensitive recommendation. In *Proc. of KDD*, 2010.

[2] D. Agarwal, B.-C. Chen, P. Elango, N. Motgi, S.-T. Park, R. Ramakrishnan, S. Roy, and J. Zachariah. Online models for content optimization. In *NIPS*, 2008.

[3] E. Agichtein, E. Brill, and S. Dumais. Improving web search ranking by incorporating user behavior information. In *Proc. of SIGIR*, 2006.

[4] R. Agrawal, S. Gollapudi, A. Halverson, and S. Ieong. Diversifying search results. In *WSDM '09*, pages 5–14, New York, NY, USA, 2009. ACM.

[5] S. Amer-Yahia, M. Benedikt, L. V. S. Lakshmanan, and J. Stoyanovich. Efficient network aware search in collaborative tagging sites. *PVLDB*, 1(1):710–721, 2008.

[6] A. Anagnostopoulos, A. Z. Broder, and D. Carmel. Sampling search-engine results. In *WWW '05*, pages 245–256, New York, NY, USA, 2005. ACM.

[7] T. Anastasakos, D. Hillard, S. Kshetramade, and H. Raghavan. A collaborative filtering approach to ad recommendation using the query-ad click graph. In *CIKM '09: Proceeding of the 18th ACM Conference on Information and Knowledge Management*, pages 1927–1930, New York, NY, USA, 2009. ACM.

[8] T. Anastasakos, D. Hillard, S. Kshetramade, and H. Raghavan. A collaborative filtering approach to sponsored search. Technical Report YL-2009-006, Yahoo! Labs, Aug 2009.

[9] X. Bai, M. Bertier, R. Guerraoui, and A.-M. Kermarrec. Toward personalized peer-to-peer top-k pprocessing. In *Workshop on Social Network Systems (SNS)*, 2009.

[10] X. Bai, M. Bertier, R. Guerraoui, A.-M. Kermarrec, and V. Leroy. Gossiping personalized queries. In I. Manolescu, S. Spaccapietra, J. Teubner, M. Kitsuregawa, A. Léger, F. Naumann, A. Ailamaki, and F. Özcan, editors, *EDBT*, volume 426 of *ACM International Conference Proceeding Series*, pages 87–98. ACM, 2010.

[11] A. Broder. Computational advertising and recommender systems. In *Proc. of the 2nd ACM International Conference on Recommender Systems (RecSys)*, 2008.

[12] G. Buscher, L. van Elst, and A. Dengel. Segmentation-level display time as implicit feedback: a comparison to eye tracking. In *Proc. of SIGIR*, 2009.

[13] J. Carbonell and J. Goldstein. The use of mmr, diversity-based reranking for reordering documents and producing summaries. In *SIGIR '98*, pages 335–336, New York, NY, USA, 1998. ACM.

[14] M. D. Choudhury, M. Feldman, S. Amer-Yahia, N. Golbandi, R. Lempel, and C. Yu. Automatic construction of travel itineraries using social breadcrumbs. In *21st ACM Conference on Hypertext and Hypermedia (HT)*, 2010.

[15] W. Chu, S. T. Park, T. Beaupre, N. Motgi, and A. Phadke. A case study of behavior-driven conjoint analysis on yahoo! front page today module. In *Proc. of KDD*, 2009.

[16] A. Das, M. Datar, A. Garg, and S. Rajaram. Google news personalization: scalable online collaborative filtering. In *Proc. of the 16th International World Wide Web Conference (WWW)*, 2007.

[17] D. Downey, S. Dumais, D. Liebling, and E. Horvitz. Understanding the relationship between searchers' queries and information goals. In *Proc. of CIKM*, 2008.

[18] G. Dupret and C. Liao. Cumulated relevance: A model to estimate intrinsic document relevance from the clickthrough logs of a web search engine. In *Proceedings of the third International ACM Conference on Web Search and Data Mining (WSDM)*, 2010.

[19] L. A. Granka, T. Joachims, and G. Gay. Eye-tracking analysis of user behavior in www search. In *Proc. of SIGIR*, 2004.

[20] Z. Guan and E. Cutrell. An eye tracking study of the effect of target rank on web search. In *Proc. of CHI*, 2007.

[21] T. Joachims, L. Granka, B. Pan, H. Hembrooke, and G. Gay. Accurately interpreting clickthrough data as implicit feedback. In *Proc. of SIGIR*, 2005.

[22] T. Joachims, L. Granka, B. Pan, H. Hembrooke, F. Radlinski, and G. Gay. Evaluating the accuracy of implicit feedback from clicks and query reformulations in web search. *ACM Transactions on Information Systems (TOIS)*, 25(2), 2007.

[23] C. Kang, S. Vadrevu, R. Zhang, R. van Zwol, L. G. Pueyo, N. Torzec, J. He, and Y. Chang. Ranking related entities for web search queries. In S. Srinivasan, K. Ramamritham, A. Kumar, M. P. Ravindra, E. Bertino, and R. Kumar, editors, *WWW (Companion Volume)*, pages 67–68. ACM, 2011.

[24] D. Kelly and N. J. Belkin. Reading time, scrolling and interaction: exploring implicit sources of user preferences for relevance feedback. In *Proc. of SIGIR*, 2001.

[25] D. Kelly and N. J. Belkin. Display time as implicit feedback: understanding task effects. In *Proc. of SIGIR*, 2004.

[26] R. Kohavi, R. M. Henne, and D. Sommerfield. Practical guide to controlled experiments on the web: listen to your customers not to the hippo. In *KDD*, pages 959–967, 2007.

[27] L. Li, W. Chu, J. Langford, and R. E. Schapire. A contextual-bandit approach to personalized news article recommendation. In *Proc. of the 19th International World Wide Web Conference (WWW)*, 2010.

[28] L. Li, W. Chu, J. Langford, and X. Wang. Unbiased offline evaluation of contextual-bandit-based news article recommendation algorithms. In *Proc. of WSDM*, 2011.

[29] C. Liu, R. W. White, and S. Dumais. Understanding web browsing behaviors through weibull analysis of dwell time. In *Proc. of SIGIR*, 2010.

[30] H. Ma, M. R. Lyu, and I. King. Diversifying query suggestion results. In M. Fox and D. Poole, editors, *AAAI '10*. AAAI Press, 2010.

[31] G. L. Nemhauser, L. A. Wolsey, and M. L. Fisher. An analysis of approximations for maximizing submodular set functions. i. *Math. Programming*, 14(3):265–294, 1978.

[32] F. Radlinski and S. Dumais. Improving personalized web search using result diversification. In *SIGIR '06*, pages 691–692, New York, NY, USA, 2006. ACM.

[33] M. Richardson, E. Dominowska, and R. Ragno. Predicting clicks: estimating the click-through rate for new ads. In *Proc. of the 16th International World Wide Web Conference (WWW)*, 2007.

[34] S. B. Roy, G. Das, S. Amer-Yahia, and C. Yu. Interactive itinerary planning. In *ICDE*, 2011.

[35] J. Teevan, C. Alvarado, M. S. Ackerman, and D. R. Karger. The perfect search engine is not enough: a study of orienteering behavior in directed search. In *Proc. of CHI*, 2004.

[36] R. W. White and S. M. Drucker. Investigating behavioral variability in web search. In *Proc. of WWW*, 2007.

[37] Z. Zheng, H. Zha, T. Zhang, O. Chapelle, K. Chen, and G. Sun. A general boosting method and its application to learning ranking functions for web search. In *Advances in Neural Information Processing Systems 20*, pages 1697–1704. MIT Press, 2008.

[38] X. Zhu, J. Guo, and X. Cheng. Recommending diverse and relevant queries with a manifold ranking based approach. In *SIGIR '10 Workshop on Query Representation and Understanding*, 2010.

IEEE Computer Society
1730 Massachusetts Ave, NW
Washington, D.C. 20036-1903

Non-profit Org.
U.S. Postage
PAID
Silver Spring, MD
Permit 1398