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

Copyright 2011 IEEE. Personal use of this material is permitted. However, permission to reprint/republish this material for advertising or promotional purposes or for creating new collective works for resale or redistribution to servers or lists, or to reuse any copyrighted component of this work in other works must be obtained from the IEEE.
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

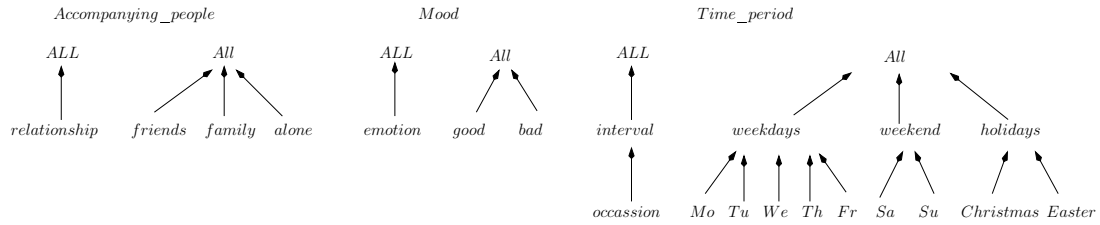

Figure 2: Hierarchy schema and concept hierarchy of *Accompanying_people*, *Mood* and *Time_period*.

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, ..., All), the graph has a single root. The preference graph is

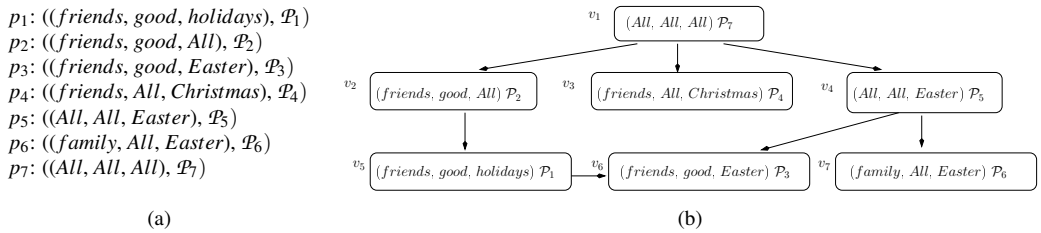

Figure 3: (a) Example set of preferences and (b) the corresponding preference graph.

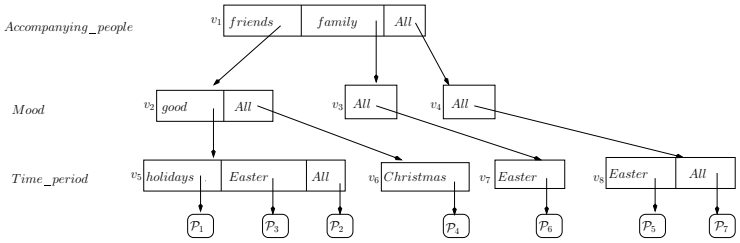

Figure 4: The profile tree for the example preferences of Fig. 3(a).

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.