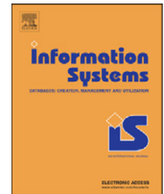
Information Systems 36 (2011) 1158–1180

Contents lists available at ScienceDirect

Information Systems

journal homepage: www.elsevier.com/locate/infosys



Managing contextual preferences

Kostas Stefanidis *, Evaggelia Pitoura, Panos Vassiliadis

Department of Computer Science, University of Ioannina, GR-45110 Ioannina, Greece

A R T I C L E   I N F O

Article history:
Received 16 February 2009
Received in revised form
1 September 2010
Accepted 12 June 2011
Recommended by: Y. Ioannidis
Available online 22 June 2011

Keywords:
Preferences
Personalization
Context

A B S T R A C T

To handle the overwhelming amount of information currently available, personalization systems allow users to specify through preferences which pieces of data interest them. Most often, users have different preferences depending on context. In this paper, we introduce a model for expressing such contextual preferences. Context is modeled using a set of hierarchical attributes, thus allowing context specification at various levels of detail. We formulate the context resolution problem as the problem of selecting appropriate preferences based on context for personalizing a query. We also propose algorithms for context resolution based on data structures that index preferences by exploiting the hierarchical nature of the context attributes. Finally, we evaluate our approach from two perspectives: usability and performance. Usability evaluates the overheads imposed on users for specifying context-dependent preferences, as well as their satisfaction from the quality of the results. Our performance results focus on the context resolution using the proposed indexes.

© 2011 Elsevier B.V. All rights reserved.

1. Introduction

Personalized information delivery aims at addressing the explosion of the amount of data currently available to an increasingly wider spectrum of users. Instead of overwhelming the users with all available data, personalization systems provide users with only the data that is of interest to them. Preferences have been used as a means to address this challenge. To this end, a variety of preference models have been proposed; most of which follow either a qualitative or a quantitative approach. With the *qualitative approach* (such as the work in [1,2]), preferences between two pieces of data are specified directly, typically using binary preference relations. For instance, using a qualitative model, users may explicitly state that they prefer visiting archaeological sites than science museums. With the *quantitative approach* (e.g., [3–5]), users employ scoring functions that associate a numerical score with specific pieces of data to indicate their interest in them. For instance, a preference in archaeological sites may be expressed by assigning high scores to such places.

However, most often users have different preferences under different circumstances. For instance, the current weather conditions may influence the place one wants to visit. For example, when it rains, a museum may be preferred over an open-air archaeological site. *Context* is a general term used in several domains, such as in machine learning and knowledge acquisition [37,39]. Our focus here is on how context can be used in conjunction with relational databases to personalize the results of queries. In this respect, we consider as context any information that can be used to characterize the situations of an entity, where an entity is a person, place, or object that is relevant to the interaction between a user and an application [6,7]. Common types of 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* [8,9].

In this paper, we propose enhancing preferences with context-related information. Preferences express user

* Corresponding author. Tel.: +30 2651098858.
E-mail addresses: kstef@cs.uoi.gr (K. Stefanidis), pitoura@cs.uoi.gr (E. Pitoura), pvassil@cs.uoi.gr (P. Vassiliadis).

0306-4379/$ - see front matter © 2011 Elsevier B.V. All rights reserved.
doi:10.1016/j.is.2011.06.004

interest on specific pieces of information stored in a relational database. There has been a variety of context models [40]. We follow a data-centric approach by representing context as a set of context parameters that take values from multi-level domains. These parameters capture information that is not part of the database, such as the *user location* or the current *weather*. A specific *context state* or situation corresponds to an assignment of values to context parameters. By allowing context parameters to take values from hierarchical domains, different levels of abstraction for the captured context data are introduced. For instance, the context parameter *user location* may take values from a *city*, *country* or *continent* domain. Preferences are enhanced with context descriptors that specify the context states under which they hold with varying levels of detail.

Each database query is also associated with one or more context states through context descriptors. The context state of a query may, for example, be the current state at the time of its submission. Furthermore, a query may be explicitly enhanced with context descriptors to allow exploratory queries about hypothetical context states. A central problem that we address in this paper is preference selection, that is, given a set of preferences and a query, determining which of the preferences are the most relevant to the query. We focus on context aspects and consider as relevant those preferences whose context states are related to those of the query. We call *context resolution* the problem of selecting preferences based on context. We consider that the context state of a preference is related to a query context state, if it is the same or more general than the query context state. This is captured through a *cover* relation defined over context states that relates context states expressed with different levels of detail. For instance, with cover, we relate a context state in which location is expressed at the level of *country* with a context state in which location is expressed at the level of *continent*. We also propose a number of distance metrics that capture similarity among context states. This allows selecting a smaller number of the qualifying preferences thus controlling the degree of personalization.

We introduce algorithms for context resolution that build upon two data structures, namely the preference graph and the profile tree, that index preferences based on their associated context states. The *preference graph* explores the partial order of context states induced by the cover relation to organize them in some form of a lattice. A top-down traversal of the graph supports an incremental specialization of a given context state, whereas a bottom-up traversal an incremental relaxation. The *profile tree* offers a space-efficient representation of context states by taking advantage of the co-occurrence of context values in preferences. It supports exact matches of context states very efficiently through a single root-to-leaf traversal.

Our focus is on managing context for preferences, i.e., expressing, storing and indexing contextual preferences. In general, preferences may be collected using various ways. Preferences may be provided explicitly by the users or constructed automatically, for instance, based on the past behavior of the same or similar users. Such methods for the automatic construction of preferences have been the focus of much current researches (e.g., [10]) and are beyond the scope of this paper. A practical way to create profiles that we have used in our experiments is to assemble a number of default profiles and then ask the users to update them appropriately.

Note that, in this paper, we take a system or designer point of view, in that, our aim is managing context efficiently. Another important aspect is to take the user point of view, for example, by expanding his interaction with the system, acquiring user feedback and supporting incremental adaptation.

We have evaluated our approach along two perspectives: usability and performance. Our *usability experiments* consider the overhead imposed on the users for specifying context-dependent preferences versus the quality of the personalization thus achieved. We used two databases of different sizes. The sizes of the database have two important implications for usability. First, they affect the number of preferences. Then, and most importantly, they require different methods for evaluating the quality of results. Our *performance experiments* focus on our context resolution algorithms that employ the proposed data structures to index preferences for improving response time and storage overheads.

In a nutshell, in this paper, we

- propose a model for annotating preferences with contextual information; our hierarchical model of context allows expressing contextual preferences at various levels of detail;
- formulate the problem of context resolution as the problem of selecting appropriate preferences for personalizing a query based on context;
- present data structures and algorithms for implementing context resolution and
- evaluate our approach in terms of both usability and performance.

The rest of this paper is structured as follows. In Section 2, we present our context and preference model, while in Section 3, we formulate the context resolution problem. In Section 4, we introduce data structures used to index contextual preferences and algorithms for context resolution. Sections 5 and 6 present our usability and performance evaluation results, respectively. Section 7 describes related work and finally, Section 8 concludes the paper with a summary of our contributions.

2. Contextual preferences

In this section, first we present our model of context, then we introduce context descriptors for specifying context states, and finally we introduce contextual preferences, e.g., preferences annotated with context information.

In the rest of this paper, we use the following two databases as our running examples.

Movie Database. The *movie* database (*MD*) maintains information about movies. It consists of a single database relation with schema *Movies* (<u>*mid*</u>, *title*, *year*, *director*, *genre*, *language*, *duration*).

Point-of-Interest Database. The *point-of-interest* database (*PID*) maintains information about interesting places to visit. It consists of a single database relation with schema *Points_of_Interest* (*pid*, *name*, *type*, *location*, *open-air*, *hours_of_operation*, *admission_cost*).

2.1. Context model

We model context using a finite set of special-purpose attributes.

Definition 1 (*Context environment*). Let X be an application. The context environment, CE_X, of X is a set of n attributes, $CE_X = \{C_1, C_2, \ldots, C_n\}$, $n \geq 1$, where each attribute C_i, $1 \leq i \leq n$, is called a context parameter.

For example, for the movie database, we consider three context parameters as relevant, namely, *accompanying_people*, *mood* and *time_period*. That is, its context environment is $CE_{MD} = \{accompanying_people, mood, time_period\}$. Preferences about movies depend on the values of these context parameters. For instance, a high preference score may be associated with movies of the genre *cartoons*, for users accompanied by their *children* and a low preference score for those accompanied by their *friends*. The context environment for the point-of-interest database is $CE_{PID} = \{user_location, weather, accompanying_people\}$. For instance, a point of interest of type *zoo* may be a more preferable place to visit than a *brewery* when accompanied by *family* and an *open-air* place like *Acropolis* a better place to visit than a *non-open-air museum*, when weather is *good*.

To allow flexibility in defining context specifications, we model context parameters as attributes that can take values with different granularities. In particular, each context parameter has multiple levels organized in a *hierarchy schema*. Let C be a context parameter with m levels, $m > 1$. 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 highest or most general one. We define a total order among the levels of each hierarchy schema L such that $L_1 \prec \cdots \prec L_m$ 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. 1 depicts the hierarchy schemas of the context parameters of our running examples. For instance, the hierarchy schema of context parameter *user_location* has four levels: *city* (L_1), *country* (L_2), *continent* (L_3) and the highest level *ALL* (L_4).

Each level L_j, $1 \leq j \leq m$, is associated with a domain of values, denoted by $dom_{L_j}(C)$. For any two levels L_j, L_k, $j \neq k$, $dom_{L_j}(C) \cap dom_{L_k}(C) = \{\}$. For all parameters, we require that their highest level has a single value *All*, i.e., $dom_{L_m}(C) = \{All\}$. We define the domain, $dom(C)$, of C as: $dom(C) = \bigcup_{j=1}^{m} dom_{L_j}(C)$. A *concept hierarchy* is an instance of a hierarchy schema. Similar to [11], a 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. 1 depicts the concept hierarchies of the context parameters of our running examples. For instance, for the context parameter *user_location*, *Greece* is a value of level *country*. Such concept hierarchies may be constructed using, for example, the WordNet [13] or other ontologies.

The relationship between the values at the different levels of a concept hierarchy is achieved through the use of a family of ancestor functions $anc_{L_j}^{L_k}$ [12],$1 \leq j < k \leq m$. The functions $anc_{L_j}^{L_{j+1}}$,$1 \leq j < m$,assign each value of the domain of L_j to a value of the domain of L_{j+1}. An edge from a node at level L_j to a node at level L_{j+1} in the concept hierarchy represents that the latter is the ancestor of the former. Given three levels L_j,L_k and L_l,$1 \leq j < k < l \leq m$,the function $anc_{L_j}^{L_l}$ is equal to the composition $anc_{L_j}^{L_k} \circ anc_{L_k}^{L_l}$. Finally, $desc_{L_l}^{L_j}(v)$,$1 \leq l < j \leq m$,gives the level l descendants of $v \in dom_{L_j}(C)$,that is, $desc_{L_l}^{L_j}(v) = \{x \in dom_{L_l}(C) | anc_{L_l}^{L_j}(x) = v\}$. For example,for the concept hierarchies in Fig. 1, $anc_{L_1}^{L_2}(Athens) = Greece$ whereas $desc_{L_1}^{L_2}(weekend) = \{Sa,Su\}$.

A context state corresponds to an assignment of values to context parameters.

Definition 2 (*Context state*). A context state cs of a context environment $CE_x = \{C_1, C_2, \ldots, C_n\}$ is an n-tuple of the form $(c_1, c_2, \ldots, c_n)$, where $c_i \in dom(C_i)$, $1 \leq i \leq n$.

For instance, (*friends*, *good*, *holidays*) and (*friends*, *All*, *summer_holidays*) 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: $CW = dom(C_1) \times dom(C_2) \times \cdots \times dom(C_n)$.

2.2. Context descriptors

Context states can be specified through context descriptors. Specifically, a single parameter context descriptor specifies values of one context parameter.

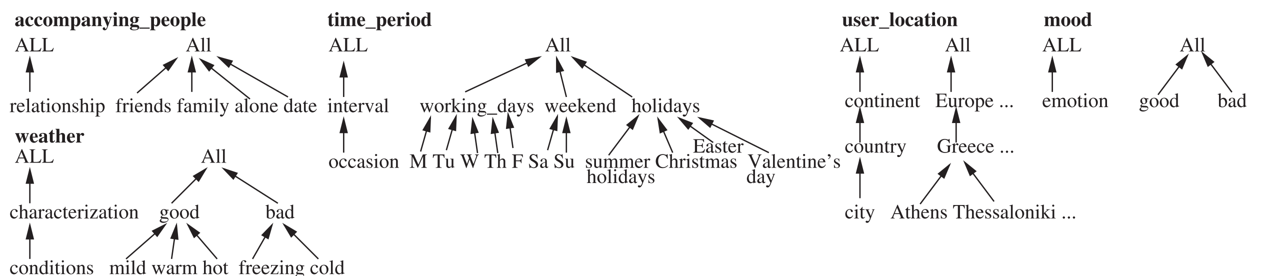


Fig. 1. Hierarchy schema and concept hierarchy of *accompanying_people*, *weather*, *time_period*, *user_location* and *mood*.

Definition 3 (*Single parameter context descriptor*). A single parameter context descriptor $cod(C)$ of a context parameter C is an expression of the form $cod(C) = C \in \{v_1, \ldots, v_l\}$, where $v_k \in dom(C)$, $1 \leq k \leq l$.

For example, for the context parameter *time_period*, a single parameter context descriptor can be *time_period* ∈ {*Christmas*} or *time_period* ∈ {*Christmas*, *Easter*, *summer_holidays*}. Let $cod(C)$ be the single context descriptor $C_i \in \{v_1, \ldots, v_l\}$, we shall use the notation $Context(cod(C_i)) = \{v_1, \ldots, v_l\}$.

Context states are specified using multi-parameter context descriptors that combine single parameter ones.

Definition 4 (*Multi-parameter context descriptor*). Let $CE_x = \{C_1, C_2, \ldots, C_n\}$ be a context environment. A multi-parameter context descriptor is an expression of the form $\bigwedge_{j=0}^{k} cod(C_{i_j})$, $1 \leq k \leq n$, where $i_j \in \{1, 2, \ldots, n\}$, $cod(C_{i_j})$ 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. These states are 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 all context parameters, we assume that the values of the absent context parameters are indifferent. In particular, if a context parameter C_i is missing from a multi-parameter context descriptor, we assume the implicit condition $C_i \in \{All\}$ to be part of the descriptor.

Definition 5 (*Context of a multi-parameter context descriptor*). Let $CE_X = \{C_1, C_2, \ldots, C_n\}$ be a context environment and $cod = \bigwedge_{j=0}^{k} cod(C_{i_j})$, $1 \leq k \leq n$, be a multi-parameter context descriptor. The set of context states of *cod*, denoted $Context(cod)$, is $\mathcal{S}_1 \times \mathcal{S}_2 \times \cdots \times \mathcal{S}_n$, where for $1 \leq i \leq n$, $\mathcal{S}_i = Context(cod(C_i))$, if $cod(C_i)$ appears in *cod* and $\mathcal{S}_i = \{All\}$, otherwise.

For the movie example, consider the multi-parameter context descriptor (*accompanying_people* ∈ {*friends*, *family*} ∧ *time_period* ∈ {*summer_holidays*}). This descriptor defines the following two context states: (*friends*, *All*, *summer_holidays*) and (*family*, *All*, *summer_holidays*).

2.3. Contextual preference model

We annotate preferences with context descriptors that specify the context states under which a preference holds. Regarding preference specification, there are, in general, two different approaches: a quantitative and a qualitative one. In the *quantitative approach* (e.g., [3]), preferences are expressed indirectly by using scoring functions that associate a numeric score or degree of interest with each item. In the *qualitative approach* (e.g., [1,2]), preferences between two items are specified directly, typically using binary preference relations. Context descriptors can be used with both a quantitative and a qualitative approach. Here, we use a simple quantitative preference model to demonstrate the basic issues underlying contextualization.

In particular, we assume that preferences for specific tuples of a database are expressed by providing a numeric score which is a real number between 0 and 1. This score expresses a degree of interest, where value 1 indicates extreme interest and value 0 indicates no interest. Interest is expressed for specific values of attributes of a database relation, for instance, for the various attributes (e.g., *genre*, *language*) of our movie database relation. This is similar to the general quantitative framework of Agrawal and Wimmers [3]. Formally, a contextual preference is defined as follows:

Definition 6 (*Contextual preference*). Given a database schema $R(A_1, A_2, \ldots, A_d)$, a *contextual preference p* on *R* is a triple (*cod*, *Pred*, *score*), where

1. *cod* is a multi-parameter context descriptor,
2. *Pred* is a predicate of the form $A_{i_1}\theta_{i_1}a_{i_1} \wedge A_{i_2}\,\theta_{i_2}\,a_{i_2} \wedge \cdots \wedge A_{i_k}\theta_{i_k}\,a_{i_k}$ that specifies conditions θ_{i_j} on the values $a_{i_j} \in dom(A_{i_j})$ of attributes A_{i_j}, $1 \leq i_j \leq d$, of *R* and
3. *score* is a real number between 0 and 1.

The meaning of such a contextual preference is that in the set of context states specified by *cod*, the database tuples that satisfy the predicate *Pred* are assigned the indicated interest *score*. In this paper, we assume that $\theta \in \{=, <, >, \leq, \geq, \neq\}$ for the numerical database attributes and $\theta \in \{=, \neq\}$ for the remaining ones. As an example, take the instance of our movie database shown in Fig. 2. The contextual preference ((*accompanying_people* ∈ {*alone*} ∧ *mood* ∈ {*bad*} ∧ *time_period* ∈ {*weekend*, *holidays*}), *genre*= *horror*, 0.8) expresses the fact that when in a *bad* mood, *alone* at a *weekend* or during a *holiday*, *horror* movies are preferred with interest score 0.8.

Note that preferences that hold irrespectively of the values of the context parameters, i.e., non-contextual preferences, may be expressed using an empty context descriptor, whose context is context state (*All*,*All*, …,*All*).

By using multi-parameter context descriptors, one can express preferences that depend on the values of more than one context parameter. Furthermore, hierarchies allow the specification of preferences at various levels of detail. For instance, one can specify preferences at the *country*, or *city* levels or both.

Finally, we define profile *P* as follows:

Definition 7 (*Profile*). Given an application *X*, a profile *P* is the set of all contextual preferences that hold for *X*.

An example profile for the movie database is shown in Fig. 3. The context $Context(P)$ of a profile *P* is the union of the contexts of all context descriptors that appear in *P*, that is, $Context(P) = \cup_i Context(cod_i)$, for each $(cod_i, Pred_i, score_i) \in P$.

mid	*title*	*year*	*director*	*genre*	*language*	*duration*
t_1	Casablanca	1942	Curtiz	Drama	English	102
t_2	Psycho	1960	Hitchcock	Horror	English	109
t_3	Schindler's List	1993	Spielberg	Drama	English	195

Fig. 2. Database instance.

$p_1 = ((accompanying_people \in \{friends\}), genre = horror, 0.8)$
$p_2 = ((accompanying_people \in \{friends\}),$
$director = Hitchcock, 0.7)$
$p_3 = ((accompanying_people \in \{alone\}), genre = drama, 0.9)$
$p_4 = ((accompanying_people \in \{alone\}),$
$(genre = drama \wedge director = Spielberg), 0.5)$

Fig. 3. Example profile.

3. Contextual preference selection

In this section, we consider the problem of selecting appropriate contextual preferences from a profile so as to personalize a given query. Our focus is on the context part. First, we define *contextual queries*. Then, given a contextual query, for a contextual preference to be selected, its context must be the same with or more general than the context of the query. This is formalized by the *cover relation* between context states that relates context states expressed at different hierarchy levels. Among such qualified candidate preferences, we select the ones whose context is the most similar to the context of the query based on two proposed distance metrics between context states. Once the appropriate preferences are selected, the query can be extended to take the selected preferences into account, as in the case of non-contextual preferences (e.g., [14,15]).

3.1. Contextual queries

Contextual queries are queries annotated with information regarding context.

Definition 8 (*Contextual query*). A contextual query *Q* is a query enhanced with a multi-parameter context descriptor denoted cod^Q which specifies its context, $Context(Q) = Context(cod^Q)$.

The context descriptor 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 contextual query corresponds to the current context, that is, the context surrounding the user at the time of the submission of the query. To capture the current context, context-aware applications use various devices, such as temperature sensors or GPS-enabled devices for location. Methods for capturing context are beyond the scope of this paper.

Besides this implicit context, we also envision queries that are explicitly augmented with multi-parameter context descriptors by the users issuing them. For example, such descriptors may correspond to exploratory queries of the form: what is a good film to watch with my *family* this *Christmas* or what are the interesting points not to be missed when I visit *Athens* with my *friends* next summer.

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. However, in some cases, it may be only possible to specify the query context using rough values, for example, when the 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.

3.2. The cover relation

Let us first consider a simple example related to the movie database. Assume a contextual query *Q* enhanced with the context descriptor $cod^Q = (accompanying_people \in \{friends\} \wedge mood \in \{good\} \wedge time_period \in \{summer_holidays\})$. If a preference with exactly the same context descriptor exists in the profile, preference selection is straightforward, i.e., this preference is selected. Assume now that this is not the case. For example, take a profile *P* that consists of three preferences: $p_1 = ((accompanying_people \in \{friends\} \wedge mood \in \{good\} \wedge time_period \in \{holidays\}), Pred_1, score_1)$ and $p_2 = ((accompanying_people \in \{friends\} \wedge mood \in \{good\} \wedge time_period \in \{All\}), Pred_2, score_2)$ and $p_3 = ((accompanying_people \in \{friends\} \wedge mood \in \{good\} \wedge time_period \in \{working_days\}), Pred_3, score_3)$. Intuitively, in the absence of an exact match, we would like to use those preferences in *P* whose context descriptor is more general than the query descriptor, in the sense that its context "covers" that of the query.

Definition 9 (*Covering context state*). A context state $cs^1 = (c_1^1, c_2^1, \ldots, c_n^1) \in CW$ covers a context state $cs^2 = (c_1^2, c_2^2, \ldots, c_n^2) \in CW$ if $\forall$ k, $1 \le k \le n$, $c_k^1 = c_k^2$ or $c_k^1 = anc_{L_i}^{L_j}(c_k^2)$ for some levels $L_i \prec L_j$.

In the example above, the context states of p_1 and p_2 cover that of *q*, whereas those of p_3 do not.

It can be shown that the cover relation imposes a partial order among context states.

Theorem 1. *The cover relation over context states is a partial order.*

Proof. We must show that the cover relation is (i) reflexive (i.e., *cs* covers *cs*), (ii) antisymmetric (if cs^1 covers cs^2 and cs^2 covers cs^1, then $cs^1 = cs^2$) and (iii) transitive (if cs^1 covers cs^2 and cs^2 covers cs^3, then cs^1 covers cs^3).

(i) Reflexivity is straightforward.
(ii) Assume for the purpose of contradiction that the antisymmetric property does not hold. In this case, there is a certain parameter C_k for which $c_k^1 = anc_{L_i}^{L_j}(c_k^2)$ and $c_k^2 = anc_{L_j}^{L_i}(c_k^1)$. But this cannot happen due to the total order of levels in a hierarchy.
(iii) Assume that cs^1 covers cs^2 (1) and cs^2 covers cs^3 (2). From (1), $\forall k$, $1 \le k \le n$, $c_k^1 = c_k^2$ or $c_k^1 = anc_{L_i}^{L_j}(c_k^2)$, $L_i \prec L_j$ (3). Respectively, from (2), $\forall k$, $1 \le k \le n$, $c_k^2 = c_k^3$ or $c_k^2 = anc_{L_i}^{L_j}(c_k^3)$, $L_i \prec L_j$ (4). Therefore, from (3), (4), we get that, $\forall k$, $1 \le k \le n$, $c_k^1 = c_k^3$ or $c_k^1 = anc_{L_i}^{L_j}(c_k^3)$, $L_i \prec L_j$, that is, cs^1 covers cs^3. □

Going back to our example, although the context states of both p_1 and p_2 cover those of the query *Q*, p_1 is more

closely related to the descriptor of the query and it is the one that should be used. Next, we formalize this notion of the most specific state or tight cover.

Definition 10 (*Tight cover*). Let P be a profile and cs^1 be a context state. We say that a context state $cs^2 \in Context(P)$ is a tight cover of cs^1 in P, if and only if:

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

In general, there may be more than one tight cover of a query context state. For example, consider again the previous query context descriptor cod^Q and assume now that P includes a fourth preference, $p_4 = ((accompanying_people \in \{friends\} \wedge mood \in \{All\} \wedge time_period \in \{summer_holidays\}), Pred_4, score_4)$. Both the context states of p_1 and p_4 are tight covers of the query context state.

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

Definition 11 (*Context resolution set*). Given a profile P and a contextual query Q, a set RS of context states, $RS \subseteq Context(P)$, is called a context resolution set for Q if (a) 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 P and (b) cs belongs to RS only if there is a $cs^Q \in Context(Q)$ for which cs is a tight cover in P.

After identifying such a set of context states, we use the contextual preferences associated with the corresponding descriptors for personalizing the query. Note that for a specific query Q and profile P, there may be no context resolution set. In this case, query Q is executed as a regular query, without using any preferences.

As shown above, for a query context state, there may be more than one tight cover. For a set of context states to qualify as a context resolution set, it must include *at least one* of them. Thus, there may be more than one context resolution sets depending on which of the tight covers of each query context state they include.

In the next section, we provide a systematic way of selecting for a given query context state which of its tight covers to include in a context resolution set by defining distances among context states. Such 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. Using too many preferences may lead to over-specializing a query, whereas using too few preferences may result in too general results. Our usability study indicates that using exactly one tight cover produces slightly more satisfying results than using more than one tight cover.

3.3. Distances between context states

To select the most appropriate among a number of tight covers, we introduce a distance metric between context states. The motivation is to choose the most specific among the candidate context states, that is, the context states defined in the most detailed hierarchy levels. We define first the level of a context state as follows.

Definition 12 (*Levels of a context state*). Let $cs = (c_1, c_2, \ldots, c_n)$ be a context state. The hierarchy levels that correspond to this state are $levels(cs) = [L_{j_1}, L_{j_2}, \ldots, L_{j_n}]$ such that $c_i \in dom_{L_{j_i}}(C_i)$, $i = 1, \ldots, n$.

The distance between two levels is defined as their path distance in their hierarchy schema.

Definition 13 (*Level distance*). Let C be a context parameter with m levels. The level distance, $dist_L(L_i, L_j)$, between two levels L_i and L_j, $1 \leq i,j \leq m$, is defined as:

$dist_L(L_i, L_j) = |j-i|$.

We can now define a level-based distance between two context states.

Definition 14 (*Hierarchy state distance*). Let $cs^1 = (c_1^1, c_2^1, \ldots, c_n^1)$ and $cs^2 = (c_1^2, c_2^2, \ldots, c_n^2)$ be two context states with $levels(cs^1) = [l_1^1, l_2^1, \ldots, l_n^1]$ and $levels(cs^2) = [l_1^2, l_2^2, \ldots, l_n^2]$. The hierarchy state distance, $dist_H(cs^1, cs^2)$, is defined as:

$$dist_H(cs^1, cs^2) = \sum_{i=1}^{n} dist_L(l_i^1, l_i^2).$$

For example, let $cs^1 =$ (*Athens*, *cold*, *alone*) be a query context state and $cs^2 =$ (*Europe*, *cold*, *alone*) and $cs^3 =$ (*Athens*, *bad*, *alone*) be two context states in the profile. In that case, $dist_H(cs^1, cs^2) = 2$ and $dist_H(cs^1, cs^3) = 1$. Both cs^2 and cs^3 cover cs^1, but cs^2 and cs^3 do not cover each other. If we assume that both cs^2 and cs^3 are tight covers of cs^1, then, using the hierarchy state distance, we would choose the preference associated with cs^3.

We show next that the hierarchy state distance produces an ordering of context states that is compatible with the cover partial order in the sense expressed by the following property.

Property 1. *Let* $cs^1 = (c_1^1, c_2^1, \ldots, c_n^1)$ *be a context state. For any two different context states* $cs^2 = (c_1^2, c_2^2, \ldots, c_n^2)$ *and* $cs^3 = (c_1^3, c_2^3, \ldots, c_n^3)$, $cs^2 \neq cs^3$, *such that* cs^2 *covers* cs^1 *and* cs^3 *covers* cs^1, *if* cs^3 *covers* cs^2, *then* $dist_H(cs^1, cs^3) > dist_H(cs^1, cs^2)$.

Proof. Let $levels(cs^1) = [l_1^1, l_2^1, \ldots, l_n^1]$, $levels(cs^2) = [l_1^2, l_2^2, \ldots, l_n^2]$ and $levels(cs^3) = [l_1^3, l_2^3, \ldots, l_n^3]$. From Definition 9, since cs^2 covers cs^1 and the fact that the level of any ancestor of c_i is larger than the level of c_i, it holds that $l_i^2 \succcurlyeq l_i^1$, $\forall i$, $1 \leq i \leq n$ (1). Similarly, since cs^3 covers cs^1, it holds that $l_i^3 \succcurlyeq l_i^1$, $\forall i$, $1 \leq i \leq n$ (2) and, since cs^3 covers cs^2, it holds that $l_i^3 \succcurlyeq l_i^2$, $\forall i$, $1 \leq i \leq n$ (3). From (1), (2) and (3), we get $l_i^3 \succcurlyeq l_i^2 \succcurlyeq l_i^1$, $\forall i$, $1 \leq i \leq n$ (4). Since $cs^2 \neq cs^3$, for at least one j, $1 \leq j$

$\leq n$, it holds that $l_j^3 \succ l_j^2$ (5). Thus, from (4), (5) and Definition 14, it holds that $dist_H(cs^1,cs^3) > dist_H(cs^1, cs^2)$. □

Property 1 states that between two context states that cover cs^1, if one of them is a tight cover, then it is the one with the smallest hierarchy state distance between them.

The context state with the minimum hierarchy state distance is not necessarily unique. For instance, assume that we want to select the context state that is most similar to $cs^1 = (friends, good, summer_holidays)$ between $cs^2 = (All, warm, holidays)$ and $cs^3 = (friends, All, All)$. For these context states, $dist_H(cs^1,cs^2) = dist_H(cs^1,cs^3) = 3$. To resolve such ties, again we choose those context states that are more specific but now in terms of the values of the detailed (lowest) level of the hierarchy that they include. The motivation is that context values that have few detailed values as descendants are more specific than those that have many such values. Clearly this is not true for all domains. But, in the absence of any other application-specific information, we assume that a value that covers many detailed values is more general than one that covers fewer ones.

For two values of two context states corresponding to the same context parameter, we measure the fraction of the intersection of their corresponding lowest level value sets over the union of these two sets and consider as a better match, the "smallest" context state in terms of cardinality. Formally, this is expressed through the Jaccard distance.

Definition 15 (*Jaccard distance*). Let C be a context parameter with m levels. The Jaccard distance, $dist_J(c_o,c_p)$, of two context values c_o and c_p, $c_o \in dom_{L_i}(C)$ and $c_p \in dom_{L_j}(C)$, $1 \leq i,j \leq m$, is defined as:

$$dist_J(c_o,c_p) = 1 - \frac{|desc_{L_1}^{L_i}(c_o) \bigcap desc_{L_1}^{L_j}(c_p)|}{|desc_{L_1}^{L_i}(c_o) \bigcup desc_{L_1}^{L_j}(c_p)|}.$$

It is easy to show that values at higher levels in the hierarchy have larger Jaccard distances than their descendants at lower levels, as the following lemma states:

Lemma 1. *Let C be a context parameter with m levels and c_o, c_p, c_q be three values of C, such that $c_o \in dom_{L_j}(C)$, $c_p \in dom_{L_k}(C)$ and $c_o \in dom_{L_l}(C)$, $L_j \prec L_k \prec L_l$, $1 \leq j,k,l \leq m$. If $c_q = anc_{L_k}^{L_l}(c_p)$ and $c_p = anc_{L_j}^{L_k}(c_o)$, then $dist_J(c_o,c_q) \geq dist_J(c_o,c_p)$.*

Proof. By definition,

$$dist_J(c_o,c_p) = 1 - \frac{|desc_{L_1}^{L_j}(c_o) \bigcap desc_{L_1}^{L_k}(c_p)|}{|desc_{L_1}^{L_j}(c_o) \bigcup desc_{L_1}^{L_k}(c_p)|}$$

and

$$dist_J(c_o,c_q) = 1 - \frac{|desc_{L_1}^{L_j}(c_o) \bigcap desc_{L_1}^{L_l}(c_q)|}{|desc_{L_1}^{L_j}(c_o) \bigcup desc_{L_1}^{L_l}(c_q)|}.$$

In both fractions, the numerator reduces to $desc_{L_1}^{L_j}(c_o)$ due to the transitivity property of the ancestor functions (i.e., all descendants of c_o at the detailed level are also descendants of c_p and c_q). The denominator of the first fraction is $desc_{L_1}^{L_k}(c_p)$, whereas the denominator of the second fraction is $desc_{L_1}^{L_l}(c_q) \supseteq desc_{L_1}^{L_k}(c_p)$, again due to the transitivity property of the ancestor function. Therefore $dist_J(c_o,c_q) \geq dist_J(c_o,c_p)$. □

The Jaccard distance between two context states is defined as follows.

Definition 16 (*Jaccard state distance*). Let $cs^1 = (c_1^1, c_2^1, \ldots, c_n^1)$ and $cs^2 = (c_1^2, c_2^2, \ldots, c_n^2)$ be two context states. The Jaccard state distance, $dist_J(cs^1,cs^2)$, is defined as:

$$dist_{JS}(cs^1,cs^2) = \sum_{i=1}^{n} dist_J(c_i^1,c_i^2).$$

For example, the Jaccard state distance of context states $cs^1 = (friends, good, summer_holidays)$ and $cs^2 = (All, All, holidays)$ is equal to: $dist_J(cs^1,cs^2) = 2$. Now, returning to our previous example for $cs^1 = (friends, good, summer_holidays)$ and the two candidates states, $cs^2 = (All, All, holidays)$ and $cs^3 = (friends, All, All)$, with the same hierarchy state distance, the following holds: $dist_{JS}(cs^1,cs^2) = 2$ and $dist_{JS}(cs^1,cs^3) = 31/22$. Therefore, cs^3 is considered to be most similar to cs^1.

It is easy to prove a property similar to Property 1, that is:

Property 2. *Let $cs^1 = (c_1^1,c_2^1,\ldots,c_n^1)$ be a context state. For any two different context states $cs^2 = (c_1^2,c_2^2,\ldots,c_n^2)$ and $cs^3 = (c_1^3,c_2^3,\ldots,c_n^3)$, $cs^2 \neq cs^3$, such that cs^2 covers cs^1 and cs^3 covers cs^1, if cs^3 covers cs^2, then $dist_{JS}(cs^1,cs^3) \geq dist_{JS}(cs^1,cs^2)$.*

Proof. Let $level(cs^1) = [l_1^1,l_2^1,\ldots,l_n^1]$, $level(cs^2) = [l_1^2,l_2^2,\ldots,l_n^2]$ and $level(cs^3) = [l_1^3,l_2^3,\ldots,l_n^3]$. From the proof of Property 1, we have that $l_i^3 \succcurlyeq l_i^2 \succcurlyeq l_i^1$, $\forall i$, $1 \leq i \leq n$. From Lemma 1, $\forall\ c_i^1, c_i^2, c_i^3$, $1 \leq i \leq n$, we get that $dist_J(c_i^1,c_i^3) \geq dist_J(c_i^1,c_i^2)$ (1), because that distance becomes larger as the context values belong to higher hierarchy levels. From (1) and Definition 16, we get that $dist_{JS}(cs^1,cs^3) > dist_{JS}(cs^1,cs^2)$. □

There may still be ties. In this case, we can randomly select any of the tight covers.

The Hierarchy and the Jaccard state distances provide a generic means for ordering tight covers. If there is additional semantic information about the context parameters and their domains, more precise distances can be defined, for example, by using weights. For instance, a weighted version of the Hierarchy state distance (Definition 14) is given by: $dist_H(cs^1,cs^2) = \sum_{i=1}^{n} w_i\ dist_L(l_i^1,l_i^2)$. where the weight w_i associated with context parameter C_i is an indication of its importance. For example, if we know that *user_location* is the determining factor for choosing a *point of interest*, we can assign weight 1 to this context parameter and 0 to the other two.

3.4. Preference application

After determining a context resolution set *RS*, the related preferences are selected for personalizing the query. In particular, the preference set $PS \subseteq P$ is formed, where $PS = \{(cod_i,Pred_i,score_i) \mid cs^j \in Context(cod_i)$ for $cs^j \in RS\}$. The preferences in *PS* can be used either (a) to reformulate the original query *Q* to include them (e.g., [5,14]) or (b) after the execution of the original query *Q*, to rank its results. A complete treatment of query personalization is beyond the scope of this paper. In the following, we focus on context-related issues. In doing so, we follow the latter approach of ordering the results of *Q*.

In particular, we rank each tuple *t* in the result *r* of *Q* based on the score of the preference in *PS* applicable to *t*. A preference $p = (cod,Pred,score)$ is applicable to a tuple *t*, if *t* satisfies predicate *Pred*. We shall use the notation $Pred[t]$ to denote that tuple *t* satisfies predicate *Pred*.

In general, more than one of the selected preferences may be applicable to a specific tuple *t* in the result *r*. In this case, we need to decide how to combine the scores of the applicable preferences for assigning a final score to *t*. Let us consider first the special case in which the predicates of the applicable preferences are related by subsumption.

Definition 17 (*Predicate subsumption*). Given two predicates $Pred_1$ and $Pred_2$, $Pred_1$ subsumes $Pred_2$, if and only if, $\forall\, t \in r$, $Pred_1[t] \Rightarrow Pred_2[t]$. In this case, we say that $Pred_1$ is more specific than $Pred_2$.

For example, take the movie relation in Fig. 2 and the profile in Fig. 3. The predicate of p_4 subsumes that of preference p_3. When a tuple *t* satisfies predicates that one subsumes the other, to compute a score for *t*, we consider only the preferences with the most specific predicates because these are considered specializations or refinements of the more general ones. In all other cases, we use the preference with the highest score, considering preferences to be indicators of positive interest.

Definition 18 (*Tuple score*). Let *P* be a profile, *cs* a context state and $t \in r$ a tuple. Let $P' \subseteq P$ be the set of preferences $p_i = (cod_i, Pred_i, score_i)$ such that $cs \in Context(cod_i)$, $Pred_i[t]$ holds and $\neg\, \exists\, p_j = (cod_j, Pred_j, score_j) \in P'$ such that $cs \in Context(cod_j)$, $Pred_j[t]$ holds and $Pred_j$ subsumes $Pred_i$. The score of *t* in *cs* is: $score(t,cs) = max_{p_i \in P'} score_i$.

For example, take context state (*friends*, *All*, *All*) and preferences p_3 and p_4. Tuple t_3 satisfies the predicates of both preferences p_3 and p_4. Since the predicate of p_4 subsumes the predicate of p_3, t_3 is assigned the score of p_4. The motivation is that p_3 expresses a degree of interest to *drama* movies in general, whereas p_4 refines p_3 by expressing a degree of interest in *drama* movies directed by *Spielberg* in particular. Since tuple t_3 is a *drama* movie directed by *Spielberg*, it is assigned the corresponding score, that is, the score of p_4. Now, take context state (*family*, *All*, *All*) and preferences p_1 and p_2 whose predicates are not related by subsumption. In this case, t_3 is assigned the largest between the two scores.

Definition 18 specifies how to compute the score of a tuple under a specific context state. However, the result of context resolution for a query *Q* may include more than one context state.

Definition 19 (*Aggregate tuple score*). Let *P* be a profile, $CS \subseteq Context(P)$ be a set of context states and $t \in r$ a tuple. The score of *t* in CS is $score(t,CS) = max_{cs \in CS} score(t,cs)$.

It is straightforward (by Definition 19) that the following holds:

Property 3. *Let cs be a context state and CS a set of context states. If* $cs \in CS$, *then for any* $t \in r$, $score(t, CS) \geq score(t, cs)$.

This means that the score of a tuple computed using a set of context states is not less than the score of the tuple computed using any of the context states belonging to this set.

Our main motivation for selecting the highest among the applicable scores is that we treat preferences as indicators of positive interest. By using the highest score, we may overrate a tuple in the result, potentially creating some form of a false positive, but we never miss any highly preferred tuple in any of the matching context states. Other choices for aggregating the applicable scores include taking the minimum or average score. Taking the minimum score corresponds to a conservative approach, where a tuple must be highly preferred in all context states. Taking the average score is a compromise between the potential overrating and underrating, respectively, caused by the maximum and minimum aggregates. However, there is no direct relation between the computed aggregated score and any of the intended scores as expressed by the applicable preferences. Note that our context resolution definition and the related algorithms are orthogonal to the selection of an aggregation method.

4. Data structures and algorithms

In this section, we focus on the efficient computation of context resolution sets. First, we consider the problem of finding the tight cover of a single query context state. One way to achieve this is by sequentially scanning all context states of all preferences in *P*. To improve response time and storage overheads, we consider indexing the preferences in *P* based on the context states in *Context*(*P*). To this end, we introduce two alternative data structures, namely the *preference graph* and the *profile tree*. We show how these structures can be used to find tight covers of a single state and compute context resolution sets. Finally, we present more efficient algorithms for locating tight covers of more than one query context state.

In the following, we call *score set* of a context state *cs* the set $W_{cs} = \{(Pred_i, score_i) \mid (cod_i, Pred_i, score_i) \in P$ and $cs \in Context(cod_i)\}$, that is, the set of predicates and the related interest scores of all preferences that include the context state *cs* in their context descriptor. A context state *cs* in *Context*(*P*) is an *exact match* of a context state cs^Q in *Context*(*Q*), if $cs = cs^Q$. If an exact match for cs^Q exists, this is clearly the unique tight cover of cs^Q in *Context*(*P*).

4.1. Preference graph

The preference graph exploits the cover relation between context states.

Definition 20 (*Preference graph*). The preference graph $PG_P = (V_P, E_P)$ of a profile P is a directed acyclic graph such that a node $v = (cs, W_{cs}) \in V_P$ for each context state $cs \in Context(P)$ and an edge $(v_i, v_j) \in E_P$, if the context state of v_i is a tight cover of the context state of v_j.

For example, for the movie database and the profile with context states shown in Fig. 4, the preference graph depicted in Fig. 5a is constructed. Note that, when there is at least one preference with context state (*All*, *All*, ..., *All*), the graph has a single root.

The preference graph is acyclic, since the cover relation over context states is a partial order (Theorem 1). The size of PG_P depends on the number of distinct context states in *P*.

Given a context state *cs* and a profile *P*, the *PG_Resolution Algorithm* (Algorithm 1) finds the states in *Context*(*P*) that are tight covers of *cs* through a top-down traversal of the preference graph PG_P of *P* starting from the nodes in V_P with no incoming edges. As its result, Algorithm 1 returns the set of nodes whose context state is a tight cover of the input context state *cs*. It also returns the Hierarchy state distance between the context state of each such node and *cs*. These distances can be used to select among the tights covers of *cs* those that are the most similar to *cs*. Search stops at a node, if it is a leaf node or if its context state does not cover *cs*. A node is included in the result only if it is a leaf node whose context state covers *cs* or the context states of all of its children do not cover *cs*.

For example, consider the preference graph in Fig. 5a and the input context state cs^Q = (*family*, *All*, *Christmas*). Search starts from the root node whose context state is cs^0 = (*All*,*All*,*All*). Since cs^0 covers cs^Q, search proceeds to nodes with context states cs^6 = (*All*, *All*, *holidays*) and cs^5 = (*family*, *All*, *All*). Both these states cover cs^Q, so the nodes with context states cs^3 = (*friends*, *All*, *holidays*) and cs^4 = (*family*, *All*, *holidays*) are visited. Context state cs^3 does not cover cs^Q, so search at this node stops, while cs^4 covers cs^Q, so the node with context state cs^2 = (*family*, *good*, *summer_holidays*) is visited. Context state cs^2 does not cover cs^Q, and since v_2 is the only child of v_4, v_4 with context state cs^4 = (*family*, *All*, *holidays*) is returned.

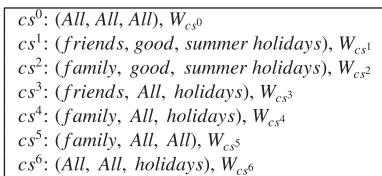

cs^0: (*All*, *All*, *All*), W_{cs^0}
cs^1: (*friends*, *good*, *summer holidays*), W_{cs^1}
cs^2: (*family*, *good*, *summer holidays*), W_{cs^2}
cs^3: (*friends*, *All*, *holidays*), W_{cs^3}
cs^4: (*family*, *All*, *holidays*), W_{cs^4}
cs^5: (*family*, *All*, *All*), W_{cs^5}
cs^6: (*All*, *All*, *holidays*), W_{cs^6}

Fig. 4. Context states with score sets.

Theorem 2 (*Correctness*). *Let PG_P be the preference graph of a profile P and Q a contextual query. If the PG_Resolution Algorithm is applied to all context states $cs^Q \in Context(Q)$, the context states of the nodes returned by the algorithm constitute a context resolution set for Q.*

Proof. The correctness of the algorithm is based on the following observation. Let cs^v be the context state of a node $v \in V_P$. Then cs^v is a tight cover of a context state cs^Q, if and only if, cs^v covers cs^Q and (i) v is a leaf node or (ii) v is an internal node and none of its children covers cs^Q. This holds because, in both cases, there is no other context state $cs \in Context(P)$ that is covered by cs^v and covers cs^Q, since if there were one, then there should be an edge in E_P from node v to the node with context state *cs*. □

Let us now discuss the complexity in the case of an exact match. Let CE_X be a context environment with n context parameters $C_1, C_2, \ldots, C_n$ with $h_1, h_2, \ldots, h_n$ levels, respectively. Let cs^Q be a query context state and cs^0 be the context state of a node with no incoming edges, i.e., one of the nodes from where search for cs^Q starts. Let us compute the maximum length of any search path from any cs^0 to cs^Q. In the worst case, all values in cs^Q belong to the most detailed hierarchy levels and cs^0 = (*All*, *All*, ..., *All*). The length of a path in this case is at most $h_1+h_2+\cdots+h_n-n$. To see this, take any edge in the search path from say a node with context state cs^i to a node with context state cs^j. In the worst case, exactly one value in cs^i is replaced in cs^j by a value of one of its immediate descendants in the corresponding concept hierarchy.

Hybrid Traversal. The *PG_Resolution Algorithm* follows a top-down approach, starting from the nodes with the most general context states and moving towards nodes with less specific context states. Alternatively, we can follow a bottom-up approach and traverse the graph starting from the nodes with the most specific context states (i.e., the leaf nodes) and move up to nodes with more general states. In this case, at each search path, search stops when the first node whose context state covers the query context state is met, since this is a tight cover. Intuitively, the bottom-up traversal is expected to outperform the top-down one for query context states that include relatively specific values, that is, for query

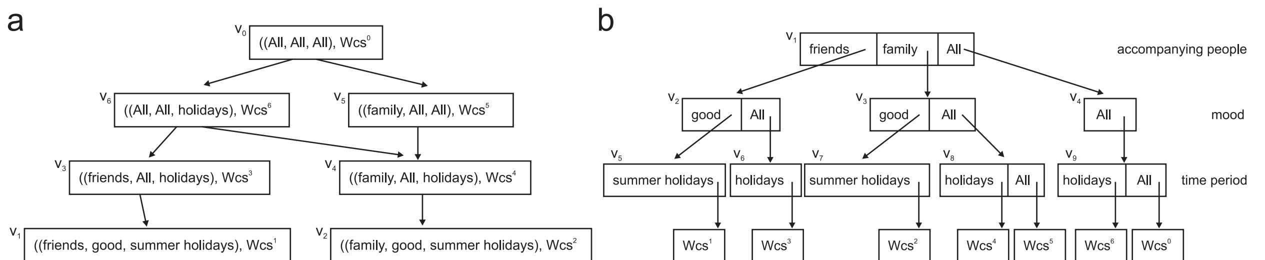


Fig. 5. An instance of (a) a preference graph and (b) a profile tree.

context states whose values belong to the lower levels of their corresponding concept hierarchies. Based on this simple observation, we consider the following heuristic for selecting the appropriate type of traversal.

We associate with each context state *cs* a *level score* $ls(cs)$ that corresponds to the average hierarchy level of its values. Specifically, let *cs* be a context state with $levels(cs) = [l_1, l_2, \ldots, l_n]$, then $ls(cs) = \sum_{i=1}^{n} l_i/n$. For example, for the context state $cs^5 = (family, All, All)$, we have $ls(cs^5) = (1+2+3)/3 = 2$. For each preference graph, we compute a level score $lp(PG_P)$ that is equal to the average level score of its nodes. Let n_G be the number of nodes of PG_P, $ls(PG_G) = \sum_{v_i \in V_P} ls(cs^i)/n_G$, where cs^i is the context state of node v_i. For example, the level score of the preference graph in Fig. 5a is 1.67.

For each query context state cs^Q, we use a top-down traversal, if $ls(cs^Q) \geq lp(n_G)$, and a bottom-up traversal, otherwise. The motivation is that if the query context state has level score greater (resp. less) than the level score of the graph, then the matching state of *cs* will probably appear high (resp. low) in the graph. For example, for the query context state $cs^{Q_1} = (family, All, holidays)$, with $ls(cs^{Q_1}) = 1.67$, a top-down traversal is used, whereas for $cs^{Q_2} = (family, good, holidays)$ with $ls(cs^{Q_2}) = 1.33$, a bottom-up traversal is used starting from nodes v_1 and v_2.

Algorithm 1. PG_Resolution Algorithm

Input: A preference graph $PG_P = (V_P, E_P)$, an input context state *cs*.
Output: A *ResultSet* of (v_i, d) pairs, such that $v_i = (cs_i, WS_i) \in V_P$, cs_i is a tight cover of *cs* and $d = dist_H(cs_i, cs)$.

```
Begin
ResultSet = ∅;
tmpV_P = ∅;
for all nodes v_i ∈ V_P do
  if v_i has no incoming edges then
   tmpV_P = tmpV_P ∪ {v_i};
  end if
  end for
while tmpV_P not empty do
   for all v_i ∈ tmpV_P do
    if cs_i covers cs then
       if v_i has no outgoing edges then
      ResultSet=ResultSet ∪ {(v_i,dist_H(cs_i,cs))};
    else
          if ∀ v_j s. t. (v_i, v_j) ∈ E_P, cs_j does not cover cs then
               ResultSet=ResultSet ∪ {(v_i,dist_H(cs_i,cs))};
                  else
               for all v_q s. t. (v_i, v_q) ∈ E_P and v_q unmarked do
                     tmpV_P = tmpV_P ∪ {v_q};
             mark v_q;
            end for
          end if
     end if
    end if
tmpV_P = tmpV_P−{v_i};
end for
end while
End
```

4.2. Profile tree

Let CE_X be a context environment with *n* context parameters C_i, $1 \leq i \leq n$. We say that a value $c \in dom(C_i)$ *appears* in a context state $cs = (c_1, c_2, \ldots, c_i, \ldots, c_n)$, if $c_i = c$. The profile tree explores any common prefixes of context states in the profile, where the *length k prefix* of $(c_1, c_2, \ldots, c_k \ldots, c_n)$ is $(c_1, c_2, \ldots, c_k)$. In particular, 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*, $t_i \in \{1, 2, \ldots, n\}$. The last level, level $n+1$, includes the leaf nodes.

Definition 21 (*Profile tree*). Let $CE_X = \{C_1, C_2, \ldots, C_n\}$ be a context environment with *n* context parameters. The profile tree T_P of a profile *P* 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 a score set.
(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 $cs \in context(P)$.
(iii) At level *k*, $1 < k \leq n$, 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 $cs \in Context(P)$ 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 $cs = (c_{t_1}, c_{t_2}, \ldots, c_{t_n})$ be the context formed by the values of the cells on the path from the root node to v_l. The leaf node v_l contains the score set W_{cs} of the context state $cs = (c_1, c_2, \ldots, c_n)$.

For example, for the movie database, and the profile with context states shown in Fig. 4, the profile tree depicted in Fig. 5b is constructed. Note that there is exactly one root-to-leaf path for each context state *cs* in *Context*(*P*). For example, the context state (*friends*, *good*, *summer_holidays*) corresponds to the path from the root to the left-most leaf node. Each leaf node maintains the score set of its corresponding context state.

The size of the profile tree T_P of a profile *P* depends on the number of common prefixes of the context states in *Context*(*P*). It also depends on the assignments of context parameters to tree levels. Let m_i, $1 \leq i \leq n$, be the cardinality of the domain of parameter C_{t_i}, that is, of the parameter assigned to tree level *i*. The maximum number of cells is $m_1 \times (1+m_2 \times (1+\cdots(1+m_n)))$. This number is as small as possible when $m_1 \leq m_2 \leq \cdots \leq m_n$, thus, in general, it is better to place context parameters that have domains with small cardinalities in the upper levels of the profile tree.

Given a context state *cs* and a profile *P*, the *PT_Resolution Algorithm* (Algorithm 2) finds the context states in *Context*(*P*) that cover *cs* through a top-down breadth-first traversal of the profile tree T_P. At each level *i*, Algorithm 2 maintains all paths of length *i* whose context state is either the same or covers the prefix $(c_{t_1}, c_{t_2}, \ldots, c_{t_i})$ of the input context state. For each candidate path, its Hierarchy state distance from the corresponding prefix of *cs* is also maintained. Algorithm 2 returns as its result the score

sets of the leaf nodes at level $n+1$ and the Hierarchy state distances of the corresponding context states from *cs*.

For example, for the profile tree of Fig. 5b and input context state $cs^Q = (family, All, Christmas)$, we start from the root node and follow the pointers of the cells containing the values *family* and *All* (i.e., the same or ancestor values of *family*) to nodes v_3 and v_4, respectively. At the next level (level 2), we follow the pointer associated with value *All*, from node v_3 to node v_8 and from node v_4 to node v_9. At the next level (level 3), we follow the pointers associated with values *holidays* and *All* at both nodes v_8 and v_9 that lead to leaf nodes with score sets W_{cs^4}, W_{cs^5} and W_{cs^6}, W_{cs^0}, respectively. Thus, the score sets of context states cs^0, cs^4, cs^5 and cs^6 are returned, which are the context states in *P* that cover cs^Q.

Lemma 2. *Let T_P be the profile tree of a profile P and cs^Q a context state. The PT_Resolution Algorithm with input T_P and cs^Q returns the set of score sets that correspond to the set of context states CR such that CR $\subseteq$ Context(P) and cs $\in$ CR if and only if cs covers cs^Q.*

Proof. Let $cs = (c_1, c_2,\ldots,c_n) \in CR$. For each i, $1 \le i \le n$, from the way the paths are formed, we have either $c_i = c_i^Q$ or $c_i = anc_{L_k}^{L_l}(c_i^Q)$, for some levels $L_k \prec L_l$. Thus, from Definition 9, if $cs \in CR$, then *cs* covers c^Q. Assume now for the purpose of contradiction that there is a $cs \in Context(P)$ such that *cs* covers cs^Q but $cs \notin CR$. Since *cs* covers cs^Q, then $\forall\, i$, $1 \le i \le n$, $c_i = c_i^Q$ or $c_i = anc_{L_k}^{L_l}(c_i^Q)$. Then, at each level *i*, the corresponding value should have been included as a candidate path and *cs* should have been in *CR*. □

Algorithm 2. PT_Resolution Algorithm

Input: A profile tree T_P, an input context state $cs = (c_1, c_2 \ldots c_n)$.
Output: A *ResultSet* of (W_{cs_i}, d) pairs such W_{cs_i} is the score set of a leaf node in T_P whose context state cs_i covers *cs* and $d = dist_H(cs_i,cs)$.

SN, *SN′*: sets of (v, d) pairs, where v is a tree node and *d* a distance value.
Initially: $SN = \{(R_P,0)\}$, where R_P is the root of T_P. $SN' = \emptyset$
anc(*c′*,*c*) returns true if value *c′* is an ancestor of value *c*
lev(*c*): the level of context value *c*
p.next: the node pointer *p* points to

```
Begin
for level i=1 to n do
  for all pairs (v, d) ∈ SN do
   for all cells (y, p) of node v do
    if y = = c_{t_i} or (anc(y,c_{t_i})) then
     if i < n then
       SN' = SN' ∪ {(p.next, d + dist_L(lev(c),lev(y))};
       else if i=n then
         W = p.next;
         ResultSet=ResultSet ∪ {(W,d)};
    end if
   end if
   endfor
  end for
 SN = SN';
  SN' = ∅;
end for
end
```

To compute the tight covers of each query context state cs^Q, $cs^Q \in Context(Q)$, we sort all returned context states for cs^Q on the basis of their Hierarchy state distance from cs^Q and select the one with the minimum such distance. If there are more than one such state, we select the context state with the smallest Jaccard state distance from cs^Q. If there are still ties, we select one of them at random. We call this algorithm, *PT_Resolution Algorithm with Sorting.*

For instance, for the covering context states of $cs^Q = (family, All, Christmas)$, computed by the *PT_Resolution Algorithm* in our previous example, the following holds: $dist_H(cs^Q,cs^0)=3$, $dist_H(cs^Q,cs^4) = 1$, $dist_H(cs^Q,cs^5) = 2$ and $dist_H(cs^Q,cs^6) = 2$. Thus, context state $cs^4 = (family, All, holidays)$ is selected as the tight cover of cs^Q.

Theorem 3 (*Correctness*). *Let T_P be the preference graph of a profile P and Q a contextual query. If the PT_Resolution Algorithm with Sorting is applied to all context states $cs^Q \in Context(Q)$, the score sets returned by the algorithm correspond to context states that constitute a context resolution set for Q.*

Proof. Let cs^1 be a context state returned by the algorithm. For the purpose of contradiction, let us assume that cs^1 is not a tight cover of any query context state cs^Q. From Lemma 2, cs^1 covers cs^Q. Since cs^1 is not a tight cover of cs^Q, there exists another context state cs^2, $cs^2 \neq cs^1$, such that $cs^2 \in Context(P)$, cs^2 covers cs^Q and cs^2 covers cs^1. From Property 1,

$$dist_H(cs^1 c^Q) > dist_H(cs^2,c^Q) \quad (1)$$

From Lemma 2, since cs^2 covers cs^Q, cs^2 is returned by the PT_Resolution Algorithm. From (1), cs^2 should have been the context state returned after sorting, which contradicts our assumption that cs^1 is returned. □

Let cs^Q be a query context state. Let us now consider the complexity of the algorithm. Assume that each context parameter C_{t_i}, $t_i \in \{1, 2,\ldots,n\}$, has h_{t_i} hierarchy levels and that there are $val(C_{t_i})$ distinct values in the profile. In the case of an exact match, i.e., that is, if there is a context state *cs* in *Profile*(*P*), such that, $cs = cs^Q$, then we can use the profile tree to locate this state very efficiently. At each level *i*, we search for value c_{t_i} and descend to the next level, following the corresponding pointer. Thus, there is just a single candidate path. At each level *i*, we just visit one node, search for the cell with value c_{t_i} in it and follow the corresponding pointer. Thus, we visit as many nodes as the height of the profile tree. At each level *i*, we need to search at most $val(C_{t_i})$ cells. Note that this can be improved, if we keep the values in the cells of each node sorted. Then, we can use binary search for locating the value c_{t_i} in the corresponding node. For the general case of looking for all covers, for each value c_{t_i}, we need to consider also its ancestors in the concept hierarchy. Thus, the number of cells that are considered for each query context state is at most $val(C_{t_1})+val(C_{t_2})\times h_{t_1}+val(C_{t_3})\times h_{t_2}\times h_{t_1}+\cdots+val(C_{t_n})\times h_{t_{n-1}}\times\cdots\times h_{t_1}$.

Enhanced Profile Tree. We present several improvements of context resolution using the profile tree. For notational simplicity, assume that each parameter C_i is mapped to level *i*, that is, $t_i = i$, $\forall\, 1 \le i \le n$. Let $cs^Q = (c_1^Q, c_2^Q,\ldots,c_i^Q, c_{i+1}^Q,\ldots, c_n^Q)$ be a query context state.

The first improvement is with regards to the test at the last internal level, i.e., level n. In this case, for the corresponding value, i.e., c_n^Q, instead of considering all values in the tree node that are equal to or more general than c_n^Q, we select the most specific among them, i.e, the value that corresponds to the lower hierarchy level. For instance, for our example query, $cs^q = (family, All, Christmas)$, at the last level, we select *holidays* and discard *All*. Now, the returned context states are cs_4 and cs_6. This improvement uses the fact that the context state formed using the most specific value is a tight cover of all other context states that would be formed by including less specific values.

The second improvement is applicable to the case of selecting just the most similar tight cover. In this case, we can reduce further the number of candidate paths maintained. In particular, we can prune those paths under construction for which there is at least another sub-path that has smaller distance to the one searched, independently of the rest of the values of the path. Assume that at level i of the tree, we have two candidate sub-paths with values $sp^1 = (sp_1^1, \ldots, sp_i^1)$ and $sp^2 = (sp_1^2, \ldots, sp_i^2)$. If for the Hierarchy state distances between the states $cs^1 = (sp_1^1, \ldots, sp_i^1, c_{i+1}, \ldots, c_n)$ and $cs^2 = (sp_1^2, \ldots, sp_i^2, All, \ldots, All)$ and the query context state cs^Q, it holds that $dist_H(cs^1, cs^Q) > dist(cs^2, cs^Q)$, then we can safely prune sub-path sp^1. The reason is that, even if, for the rest of the values of sp^1, we could find in the tree values equal to that of the query (best case scenario), its distance from the context query state would still be greater than that of sp^2 even if we could not find anything better than *All* for the remaining values of sp^2 (worst case scenario).

Finally, to speed up context resolution, we add cross edges, called *hierarchical pointers*, among context values that belong to a specific node. In particular, we link each value with its first ancestor that exists in the node, that is, the value that belongs to the first upper level of the corresponding hierarchy. For instance, for the profile tree of Fig. 5b, we add cross edges from *good* to *All* at level *mood* and edges from *holidays* to *All* at level *time period*. The values within each node are sorted according to the hierarchy level to which they belong. Now, instead of searching within each node for the more general values of each context value, we just follow the hierarchical pointers of the tree to locate them.

4.3. Resolution for multiple context states

In the previous sections, we have used the preference graph and the profile tree to locate tight covers of each context state $cs^Q \in Context(Q)$ individually. In this section, we propose algorithms for locating tight covers of more than one query context state. The idea is to use a representation corresponding to a preference graph or profile tree for $Context(Q)$. The only difference is that there are no score sets.

4.3.1. Using the preference graph

Let us consider first a preference graph based representation of $Context(Q)$.

Definition 22 (*Query graph*). The query graph $G_Q = (V_Q, P_Q)$ of a contextual query Q is a directed acyclic graph with a node $v \in V_Q$ for each context state $cs \in Context(Q)$ and an edge $(v_i, v_j) \in P_Q$, if the context state of v_i is a tight cover of the context state of v_j.

Our approach is based on the following observation.

Lemma 3. *Let P be a profile and cs^1 and cs^2 be two context states such that cs^1 covers cs^2. Let $cs^z \in Context(P)$ be a tight cover of cs^1 in P and let z be the corresponding node in the preference graph PG_P of P. Then, none of the predecessors of z in PG_P corresponds to a context state that is a tight cover of cs^2.*

Proof. For the purpose of contradiction, assume that there is a node v with context state cs^v in PG_P such that v is a predecessor of z and its context state cs^v is a tight cover of cs^2. Since v is a predecessor of z, cs^v covers cs^z (1). Since cs^z covers cs^1 and cs^1 covers cs^2, cs^z covers cs^2 (2). From (1), (2), cs^v cannot be a tight cover of cs^2. □

We use Lemma 3 as follows. Let cs^Q be a query context state of a node in G_Q with no incoming edge. We form a set $S(cs^Q) \subset Context(Q)$ with the context states that cs^Q covers. Assume that we use *the PG_Resolution Algorithm* to locate a tight cover of cs^Q in P, say the context state of node v. Then, when looking for tight covers for the context states in $S(cs^Q)$, we can ignore all predecessors of v in PG_P. We use the query graph G_Q to compute $S(cs^Q)$. From the definition of the query graph, a context state $cs \in S(cs^Q)$, if and only if, there exists a path in G_Q from the node with context state cs^Q to the node with context state cs.

We further improve the above procedure by processing nodes in $S(cs^Q)$ gradually as follows. We first locate all context states in $Context(Q)$ that are tight covers of cs^Q (these are the context states for which there is an edge in G_Q from the node that corresponds to context state cs^Q to their corresponding node). Let cs be such a context state. We use the *PG_Resolution Algorithm* to locate a tight cover of cs excluding the predecessors of the tight covers of cs^Q. Then, we proceed similarly for cs. That is, we find all tight covers of cs and use the *PG_Resolution Algorithm* to locate a tight cover for them excluding the predecessors of the tight covers of cs. This procedure repeats until we locate tight covers of all context states in $Context(P)$. We call this algorithm, *QG_Resolution Algorithm*.

4.3.2. Using the profile tree

The context states in $Context(Q)$ are represented by a data structure similar to the profile tree, that we call *query tree*. The only difference is that the leaf nodes are empty; there is no score set associated with the root-to-leaf paths. The mapping of the context parameters to the levels of the query tree is the same with the mapping of context parameters to the levels of the profile tree. For simplicity, assume that $\forall i$, $1 \le i \le n$, context parameter C_i is mapped to level i of both the profile and the query tree.

The *QT_Resolution Algorithm* (Algorithm 3) processes pairs of nodes that belong to the same level, i.e., values

that belong to the same context parameter. Each pair consists of a node of the query tree and a node of the profile tree. Initially, there is one pair of nodes, (R_Q, R_P, 0), where R_Q and R_P are the root nodes of the query and the profile tree, respectively, (level $i=1$). For each value in any cell of the query node R_Q that is equal to a value in any cell of the profile node R_P or belongs to a lower hierarchy level, we create a new pair of nodes at the next level ($i+1$). After checking all values of all pairs at a specific level, we examine the pairs of nodes created for the immediately next level and so on. At level $n+1$, we retrieve from the profile tree the associated score set. Observe that the *QT_Resolution Algorithm* tests for all query states in a single pass of the profile tree.

Algorithm 3. QT_Resolution Algorithm

Input: A *profile tree* T_P, a query tree T_Q
Output: A *ResultSet* of (W_{cs},d) pairs such that W_{cs} is the score set of a leaf node in T_P whose context state *cs* covers at least one context state cs^Q in *Context*(Q) and $d = dist_H(cs,cs^Q)$.

SN, *SN′* sets of (v_q,v_p,d) tuples, where v_q is a node in T_Q, v_p is a node in T_P and d a distance value.
Initially: $SN=\{(R_Q,R_P,0)\}$, where R_Q, R_P are the root nodes of T_Q and T_P, respectively. $SN'=\emptyset$
anc(c',c), *lev*(c), *p.next* as in Algorithm 2

```
Begin
for level i=1 to n do
 for all tuples (v_q,v_q,d) ∈ SN do
   for all cells (x, p_q) of node v_q do
    for all cells (y, p_p) of node v_q do
     if x = = y or (anc(y,x)) then
      if i < n then
       SN' = SN' ∪ {(p_q.next, p_p.next, d+dist_L(lev(x),lev(y)))};
    else if i=n then
       W = p_p.next;
       ResultSet=ResultSet ∪ {(W,d)};
         end if
      end if
    end for
   end for
  SN = SN';
 SN' = ∅;
end for
End
```

Lemma 4. *Let T_P be the profile tree of a profile P and T_Q the query tree of a query Q. The QT_Resolution Algorithm returns the set of score sets that correspond to the set of context states CR such that $CR \subseteq Context(P)$ and (i) $cs \in CR$, if and only if, cs covers cs^Q, $cs^Q \in Context(Q)$ and (ii) $\forall\, cs^Q \in Context(Q)\ \exists\, cs \in CR$, such that cs covers cs^Q.*

Proof. As in Lemma 2, the proof follows from the fact that all pairs of values of the corresponding context parameters are considered by the algorithm. □

5. Usability evaluation

The goal of our usability study is to justify the use of contextual preferences. In particular, the objective is to show that, for a reasonable effort of specifying contextual preferences, users get more satisfying results than when there are no preferences or when the available preferences do not depend on context.

We used two databases of different sizes: (a) a relatively small point-of-interest database and (b) a relatively large movie database. The point-of-interest database consists of nearly 1000 real points-of-interest of the two largest cities in Greece, namely, Athens and Thessaloniki. The context parameters are *accompanying_people*, *time_period* and *user_location*. For the movie database, our data comes from the Stanford Movie Database [16] with information about 12 000 movies. The context parameters are *accompanying_people*, *mood* and *time_period*. Note that the context parameters are the same as those used in our running examples with the exception of *time_period* used in place of *weather* in the point-of-interest database.

The sizes of the databases have two important implications for usability. First, they affect the size of the profile, i.e., the number of preferences. Second, they require different methods for evaluating the quality of results. Specifically, while for the small point-of-interest database, we can ask users to manually provide the best results, for the large movie database, we need to use other metrics [17].

We conducted an empirical evaluation of our approach with 20 users; 10 different users were used for each of the two databases. For all users, it was the first time that they used the system. We evaluated our approach along two lines: overhead of profile specification and quality of results.

5.1. Profile specification

To ease the specification of preferences, we created a number of default profiles for each database based on three characteristics: (a) age (below 30, between 30 and 50, above 50), (b) sex (male or female) and (c) taste (broadly categorized as mainstream or out-of-the-beaten track). For each of the 12 possible combinations of the values of the above characteristics, we created one profile with contextual-preferences and one profile with non-contextual preferences, i.e., preferences that hold independently of the values of the context parameters. One contextual and one non-contextual profiles were pre-assigned to each user based on his/her age, sex and taste. Users were allowed to modify the default profiles assigned to them by adding, deleting or updating preferences.

We counted the number of modifications (insertions, deletions, updates) of preferences of the default profile. We also reported how long (in minutes) it took users to specify/modify their profile. Since for all users this was their first experience with the system, the reported time includes the time it took the user to get accustomed with the system. These results are reported in Table 1 for points-of-interest and Table 2 for movies, while Table 3 summarizes them. For the point-of-interest database, each default non-contextual profile has about 100 preferences, while each default contextual profile nearly 650 preferences, while for the movie database, the sizes are 120 and 1100, respectively.

The general impression is that predefined profiles save time in specifying user preferences. Furthermore, having default profiles makes it easier for someone to understand

Table 1
Point-of-interest dataset: overhead of profile specification per user.

Profiles	User 1 (%)	User 2 (%)	User 3 (%)	User 4 (%)	User 5 (%)	User 6 (%)	User 7 (%)	User 8 (%)	User 9 (%)	User 10 (%)
Non-contextual profile										
Number of updates	15	14	8	11	15	16	19	12	10	10
Update time (min)	13	8	5	6	6	9	16	7	9	8
Contextual profile										
Number of updates	22	31	12	28	24	32	38	13	18	25
Update time (min)	30	45	20	30	30	40	45	15	20	25

Table 2
Movie dataset: overhead of profile specification per user.

Profiles	User 1 (%)	User 2 (%)	User 3 (%)	User 4 (%)	User 5 (%)	User 6 (%)	User 7 (%)	User 8 (%)	User 9 (%)	User 10 (%)
Non-contextual profile										
Number of updates	37	14	29	10	22	31	29	15	17	12
Update time (min)	18	7	14	6	9	19	16	6	8	8
Contextual profile										
Number of updates	67	49	52	91	37	72	69	41	46	37
Update time (min)	32	28	28	55	17	44	36	22	27	46

Table 3
Average profile specification overhead.

Profiles	Point-of-interest dataset	Movie dataset
Non-contextual profile		
Number of updates	13	21.6
Update time (min)	8.7	11.1
Contextual profile		
Number of updates	24.3	56.1
Update time (min)	30	33.5

the main idea behind the system, since the preferences in the profile act as examples. With regards to time, there was deviation among the time users spent on specifying profiles: some users were more meticulous than others, spending more time in adjusting the profiles assigned to them. As expected, the specification of contextual profiles is more time-consuming than the specification of non-contextual ones, since such profiles have a larger number of preferences and are more fine-grained. The size of the database also affects the complexity of profile specification basically by increasing the number of preferences and thus, the required modifications. However, the increase in the total time spent is not necessarily proportional to the number of modifications, since this time, as explained, also includes the time to get acquainted with the system (Table 3).

5.2. Quality of results

In this set of experiments, our goal is to evaluate the quality of the results of contextual queries. Queries were executed: (i) without using any of the preferences, (ii) using the non-contextual preferences and (iii) using the contextual preferences. When using contextual preferences, we consider queries for which: (iii-a) there is an exact match, (iii-b) there is no exact match and the most similar (top-1) tight cover is used and (iii-c) there is no exact match and the three most similar (top-3) tight covers are used. For computing similarity, we used the Hierarchy state distance, to resolve ties, the Jaccard state distance and if there were still ties, random selection.

We asked the users to evaluate the quality of the results. Since the point-of-interest database has a small number of tuples, we asked the users to rank the results of each contextual query manually. Then, we compare the ranking specified by the users with what was recommended by the system. We report the percentage of the top-20 results computed by the system that also belonged to the top-20 results given by the user, or *precision*(*20*). As shown in Table 4, this percentage is generally high. However, sometimes the choices of the user did not conform even to their own preferences as shown in the case of queries with an exact match. In this case, although the context state of the preference used was an exact match of the context state of the query, still some users ranked their results differently than what the related preference indicated. Note that in general, users who customized their profile by making more modifications (e.g., User 6 in Table 1) got more satisfactory results than those that spent less time during profile specification (e.g., User 3 in Table 1). Exact match queries provide the best results, while non-exact match ones provide only slightly worse results. That is, if there are no preferences whose context state is equal to that of the query, preferences whose context state is a tight cover can be used. Furthermore, for such non-exact match cases, using just the most similar (top-1) tight cover provides slightly better results than using more (top-3) similar tight covers. Since using just one cover is also more efficient, this seems to be the best choice for a context resolution set.

For the movie database, due to the large number of tuples, it was not possible for the users to manually rank

Table 4
Point-of-interest dataset: quality of results per user.

Personalization method	User 1 (%)	User 2 (%)	User 3 (%)	User 4 (%)	User 5 (%)	User 6 (%)	User 7 (%)	User 8 (%)	User 9 (%)	User 10 (%)
No preferences	10	0	0	0	0	10	5	0	0	5
Non-contextual preferences	10	10	0	5	5	10	15	5	5	5
Contextual preferences										
Exact match	100	90	90	95	90	100	100	85	100	100
Non-exact match										
Top-1 tight cover	100	95	90	85	90	100	100	85	90	100
Top-3 tight covers	95	90	85	95	95	90	100	75	85	95

Table 5
Movie dataset: quality of results per user.

Personalization method	User 1 (%)	User 2 (%)	User 3 (%)	User 4 (%)	User 5 (%)	User 6 (%)	User 7 (%)	User 8 (%)	User 9 (%)	User 10 (%)
No preferences										
Precision(20)	25	20	35	35	20	30	20	35	15	30
Highly preferred movies	10	0	10	15	15	10	10	20	15	5
Overall score	3	3	3	2	4	3	3	3	2	1
Non-contextual preferences										
Precision(20)	20	30	40	60	60	60	35	30	40	25
Highly preferred movies	0	5	25	20	30	35	20	15	25	15
Overall score	2	3	4	6	6	6	4	3	4	2
Contextual preferences										
Exact match										
Precision(20)	75	85	75	65	85	70	85	85	90	85
Highly preferred movies	70	70	75	60	65	70	75	75	85	70
Overall score	8	9	8	8	7	8	8	8	9	9
Non-exact match										
Top-1 tight cover										
Precision(20)	75	85	65	55	65	65	70	85	85	70
Highly preferred movies	50	70	60	45	40	60	65	60	75	55
Overall score	6	9	7	7	6	8	8	8	8	7
Top-3 tight covers										
Precision(20)	65	75	60	55	65	60	70	85	85	70
Highly preferred movies	45	65	55	50	45	40	55	55	70	50
Overall score	6	8	7	7	6	6	7	8	8	7

all results. Instead, users were asked to evaluate the quality of the 20 higher ranked movies in the result. For characterizing the quality of the results, users marked each of the 20 movies with 1 or 0, indicating whether they considered that the movie should belong to the best 20 ones or not, respectively. The number of 1 s corresponds to the precision of the top-20 movies, or *precision(20)*. Furthermore, users were asked to give a specific numerical interest score between 1 and 10 to each of the 20 movies. This score was in the range [1, 5], if the previous relevance indicator was 0 and in the range [6, 10], otherwise. We report the number of movies that were rated highly (interest score $\geq$ 7). Finally, users were asked to provide an overall score in the range [1, 10] to indicate their degree of satisfaction of the overall result set. Table 5 depicts the detailed per user scores attained for the movie database. Again, our results show that using contextual preferences improves quality considerably.

When compared to the point-of-interest database (Table 6), precision is lower. One reason for that is the following. In the movie database, the users were not aware of the whole result set; they were just presented with the top 20 movies in the result. Thus, they "left room" in their choices for better results that could be lying in the dataset that was not presented to them.

6. Performance evaluation of preference selection

In the section, we present an experimental evaluation of preference selection using the proposed data structures, namely the preference graph, the profile tree and their enhancements. We used both real and synthetic profiles. As real profiles, we used the ones specified by the users in our usability study for the movie and the point-of-interest databases.

We have generated synthetic profiles with various characteristics and sizes. We consider preferences with 2, 3 and 4 context parameters. Context parameters have domains with different cardinalities, namely, a small domain with 10 values, a medium domain with 100 values and a large domain with 1000 values. Small and medium domains correspond to context parameters such

Table 6
Average quality of results.

Personalization method	Point-of-interest dataset	Movie dataset		
	Precision (20) (%)	Precision (20) (%)	Highly preferred movies (%)	Overall score
No preferences	3	26.5	11	2.7
Non-contextual preferences	7	40	19	4
Contextual Preferences				
Exact match	95	80	71.5	8.2
Non-exact match				
Top-1 tight cover	93.5	72	58	7.4
Top-3 tight covers	90.5	69	53	7

Table 7
Input parameters for synthetic profiles.

Parameter	Range	Default
Number of contextual preferences	500–10 000	5000
Number of context parameters	2, 3, 4	3
Data distribution	uniform, zipf	
	$a=0-2$	$a=1$
Cardinality of context domains	10, 100, 1000	
Hierarchy levels	2–8	4
Perc. of values at the detailed level	75–25	75
Perc. of values at the other levels	25–75	25

as *mood* or *accompanying_people*, whereas larger domains to parameters such as *user_location* or *time_period*. We also consider hierarchies with anywhere between 2 and 8 levels and different distributions of the domain values among the levels. Since the preference graph and the profile tree take advantage of co-occurrences of context states and prefixes of context states, respectively, their size depends on the distribution of context values in the context states that appear in the profile. To populate a profile with context states, we consider for the context values both a uniform and a zipf data distribution with different values of a. The context states for the queries are generated similarly. Table 7 summarizes the parameters used for creating the synthetic profiles.

We report results regarding (a) the size of the corresponding data structures and (b) the complexity of preference selection using the proposed data structures. The results are averaged over 50 executions.

6.1. Storage

In this set of experiments, we evaluate the space requirements (in number of cells) for storing context states using the profile tree and the preference graph as opposed to storing them sequentially (no index). For the profile tree, this depends on the mapping of context parameters to tree levels. Thus, we created profile trees for all possible mappings of context parameters to levels of the trees.

Synthetic data. First, we consider profiles of different sizes, that is, with different numbers of contextual preferences. We created a profile tree for all six different mappings of parameters to tree levels. Let S stand for the small, M for the medium and L for the large domain. We use (S,M,L) to denote the mapping in which S is assigned to the first level of the tree, M to the second and L to the third one. Similarly, we use (S,L,M), (M,S,L), (M,L,S), (L,S,M) and (L,M,S) to denote the remaining mappings. The mapping of parameters with large domains lower in the tree results in smaller trees as expected (Figs. 6a,b). In the following experiments, we use this mapping for the profile tree, unless specified otherwise. For a zipf distribution with $a=1$ (Fig. 6a), for both the profile tree and the preference graph, the total number of cells is smaller than that for the uniform distribution (Fig. 6b), because "hot" values appear more frequently in preferences, i.e., more preferences have overlapping context states. Overall, the profile tree is smaller than the preference graph, since it takes advantage of repetitions of prefixes of context states, whereas the preference graph considers repetitions only of whole context states.

We also keep the size of the profile fixed to its default value and vary the values of the other parameters. In Fig. 7a, we present results for various values of a; the larger the value of a, the larger the difference between the size of the profile tree and the preference graph, since a larger number of overlapping prefixes is created. Fig. 7b depicts our results for profiles with different numbers of context parameters, specifically for 2 parameters with an M and an L domain, for three parameters and an S, M and an L domain and for four parameters with an S, M, M and an L domain. As expected, the size of the data structures increases with the number of the context parameters, since the number of overlapping context state and prefixes reduces. Note that the size of the data structures does not depend on the assignments of context values to hierarchy levels or the number of hierarchy levels.

Real data. For the movie database, let A stand for *accompanying_people*, M for *mood* and T for *time_period*. As before, we use (A,M,T), (A,T,M), (M,A,T), (M,T,A), (T,A,M) and (T,M,A) for the possible mappings of parameters to tree levels. Accordingly, for the point-of-interest database, let A stand for *accompanying_people*, T for *time_period* and L for *user_location*. Then, *(A, T, L)*, *(A, L, T)*, *(T, A, L)*, *(T, L, A)*, *(L, A, T)* and *(L, T, A)* denote the different mappings of parameters to tree levels. As shown in Fig. 8, both the preference graph and the profile require less space than storing preferences sequentially, since each

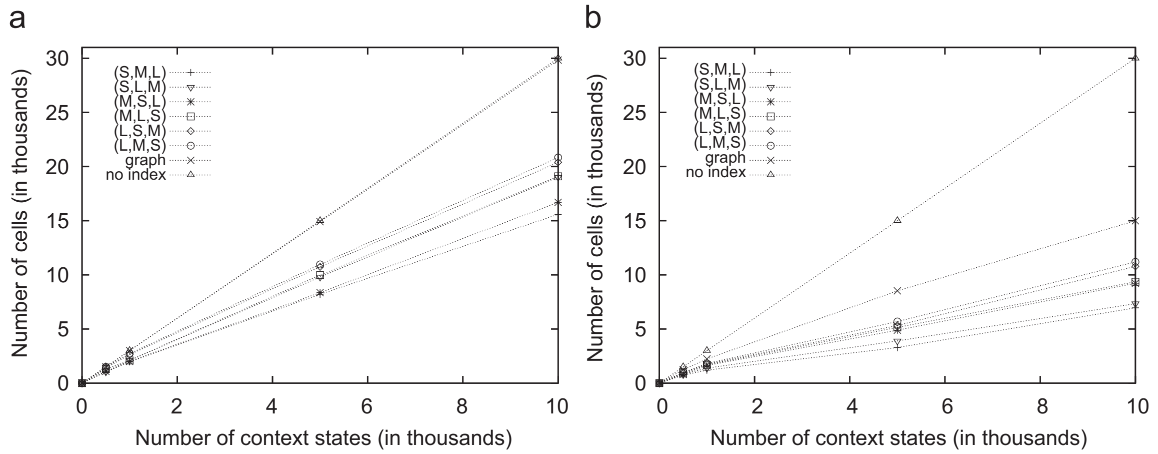


Fig. 6. Size for synthetic profiles of different sizes with (a) uniform and (b) zipf with $a=1$ data distributions.

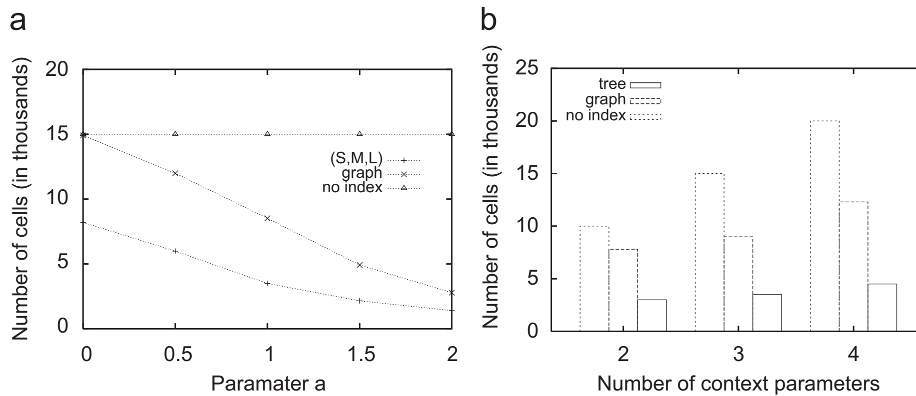


Fig. 7. Size for synthetic profiles generated with different (a) a values and (b) numbers of context parameters.

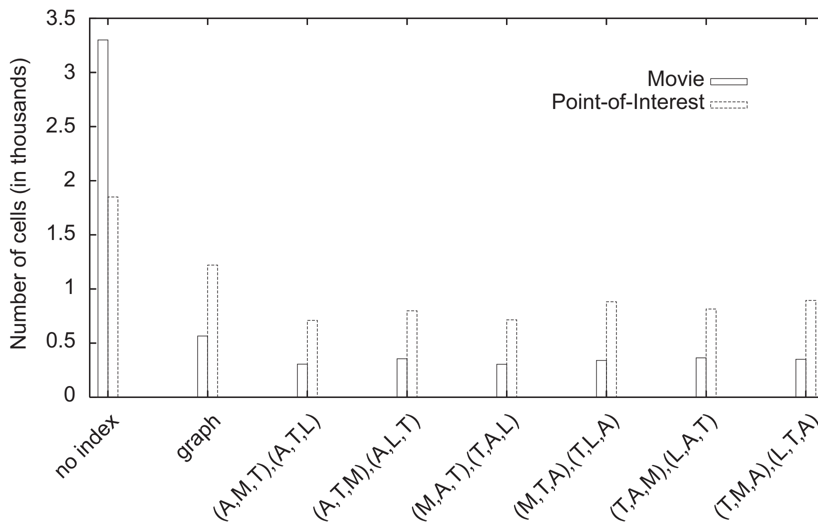


Fig. 8. Size for the case of real profiles.

context state and, respectively, prefix is stored only once. For the profile tree, the mappings that result in trees with smallest sizes are, as expected, the ones that map the context parameters with large domains to levels lower in the tree, namely, mappings *(A, M, T)* and *(M, A, T)* for the movie database and mappings *(A, T, L)* and *(T, A, L)* for the

point-of-interest database. However, all trees occupy less space than storing preferences sequentially.

6.2. Preference selection

In this set of experiments, we evaluate the performance of preference selection (in term of cell accesses). This depends on whether there is a context state in the profile that is exactly the same as the query context state. Thus, we study these two cases (i.e., exact and non-exact match) separately. In the case of sequential scan (i.e., no index), for exact matches, the profile is scanned until the matching context state is found, while for non-exact matches, the whole profile is scanned. With the profile tree, exact match queries are resolved by a simple root-to-leaf traversal, while for non-exact matches, multiple candidate paths are maintained.

We consider first a single query context state. We evaluate the performance of preference selection for different profile sizes (Figs. 9a,b). Observe 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. The cost of sorting is not reported in these experiments. Note that sorting is only required in the case of non-exact matches. In general, excluding sorting, the profile tree is more efficient than the preference graph. The preference graph performs similarly for both exact and non-exact matches. In Fig. 9c, we also evaluate the heuristic for the preference graph. In particular, we compare the top-down, bottom-up and hybrid traversal. Overall, the bottom-up approach requires more cell accesses than the top-down approach, since in general, there are more nodes at the lower levels than in the upper ones. In the following, we use the enhanced profile tree and the hybrid traversal for the preference graph.

Besides the profile size, we also consider how the other parameters affect the performance. In Fig. 10, we consider a zipf distribution with different a values for exact (Fig. 10a) and non-exact (Fig. 10b) matches. Larger a values result in fewer cell accesses, since the resulting data structures are smaller. Fig. 11 reports results when different percentages of context values are assigned to hierarchy levels; the larger the number of values at the higher levels, the smaller the number of cell accesses. The reason is that, in this case, for the preference graph,

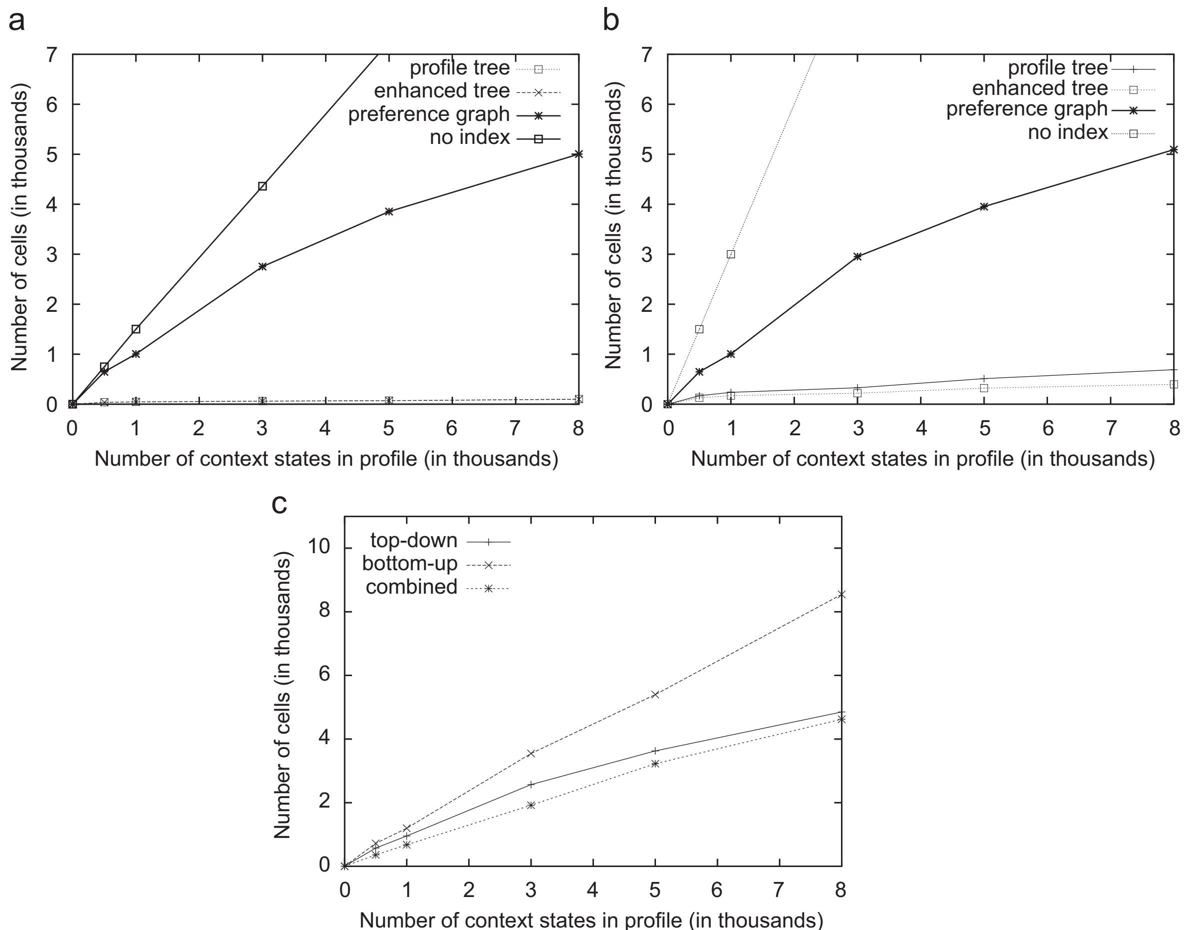


Fig. 9. Cell accesses for locating preferences related to queries using the profile tree, the enhanced profile tree and the preference graph, and when no index is used, for synthetic profiles for (a) exact match and (b) non-exact match and (c) for the top-down, bottom-up and the heuristic approach in the case of the preference graph for non-exact match (results for exact match are similar for the graph).

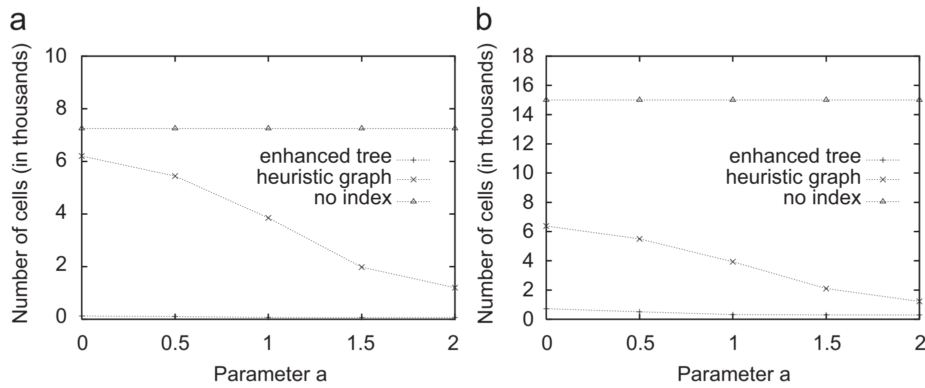


Fig. 10. Cell accesses for locating preferences related to queries for synthetic profiles generated with different *a* values for (a) exact match and (b) non-exact match.

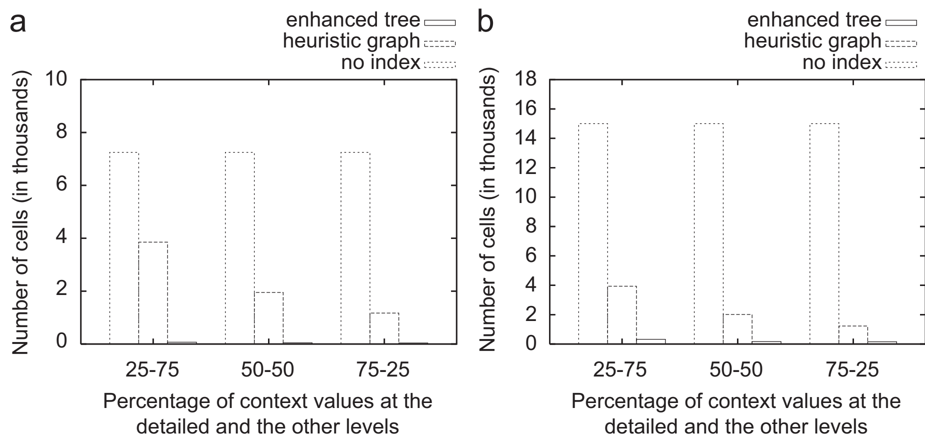


Fig. 11. Cell accesses for finding preferences related to queries for synthetic profiles having different percentages of context values between the detailed and the other hierarchy levels for (a) exact match and (b) non-exact match.

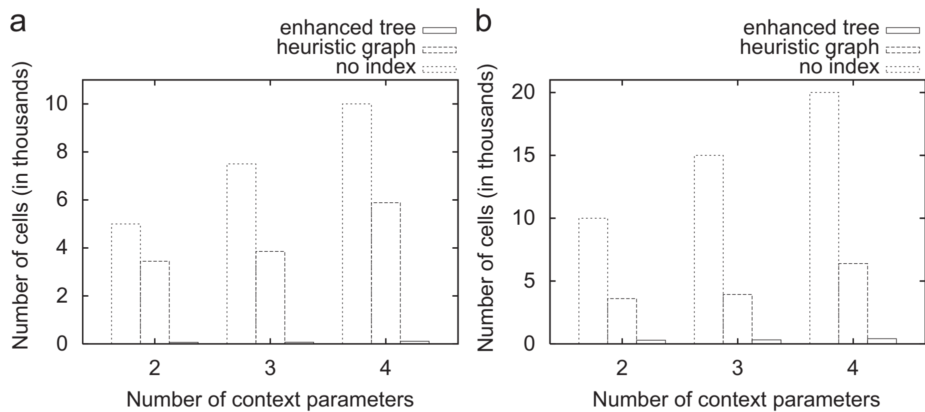


Fig. 12. Cell accesses for locating preferences related to queries for synthetic profiles with different numbers of context parameters for (a) exact match and (b) non-exact match.

context resolution stops at higher levels and for the profile tree, there are less covering context states and thus candidate paths. Fig. 12 shows results for profiles with different numbers of context parameters. We use profiles with 2, 3 and 4 context parameters with domains as in Fig. 7b. The number of cell accesses increases as the number of context parameters increases, mainly, because the size of the data structures increases. The number of hierarchy levels affects only non-exact matches (Fig. 13). In this case, more cell accesses are required, since we search for additional, more general context values; exact matches do not directly depend on the number of levels.

The results for the real profiles are depicted in Fig. 14. Overall, again the profile tree provides the most efficient preference selection, if we ignore the overhead of sorting.

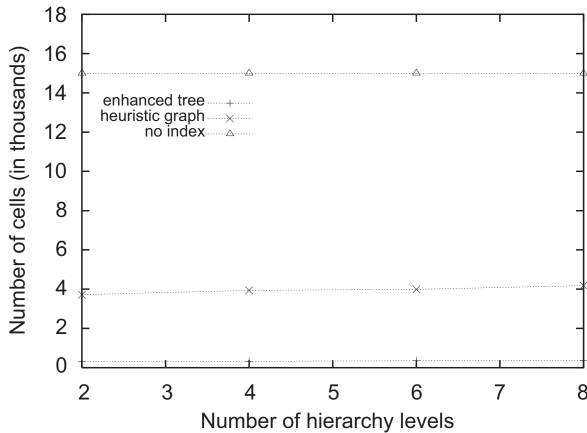


Fig. 13. Cell accesses for finding preferences related to queries for synthetic profiles with different numbers of hierarchy levels for non exact match.

Next, we compare the performance of searching for more than one matching context states for queries with varying number of context states using the *QT_Resolution* and the *QG_Resolution* algorithms. Fig. 15 depicts our results for exact (Fig. 15a) and non-exact (Fig. 15b) match queries. Both optimizations result in more efficient resolution than considering each context state individually.

The results for the real profiles are shown in Fig. 16. We run this experiment for exact match queries (Fig. 16a) and for non-exact ones (Fig. 16b), with query descriptors consisting of 20 context states. Using the proposed algorithms reduces access time in all cases.

7. Related work

In this paper, we use context to confine database querying by selecting as results the best matching tuples based on user preferences that depend on context. We first review research on preferences, then on context and finally, on contextual preferences.

The research literature on preferences is extensive. In particular, in the context of database queries, there are

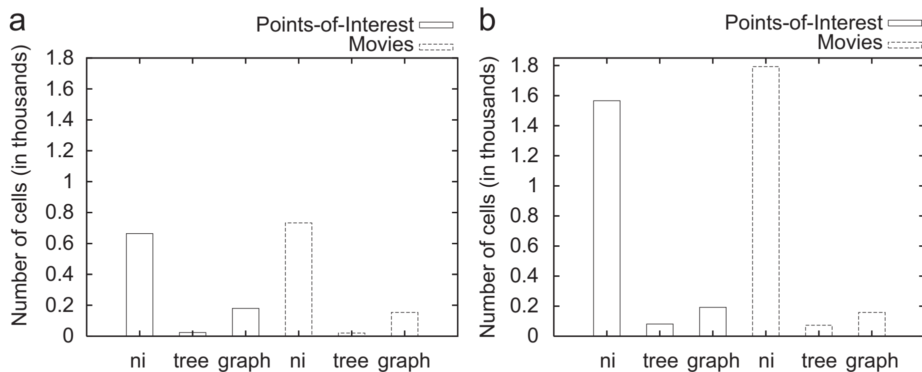


Fig. 14. Cell accesses for finding preferences related to queries using the enhanced profile tree (denoted with *tree*), the heuristic approach for the preference graph (denoted with *graph*), and when no index is used (denoted with *ni*).

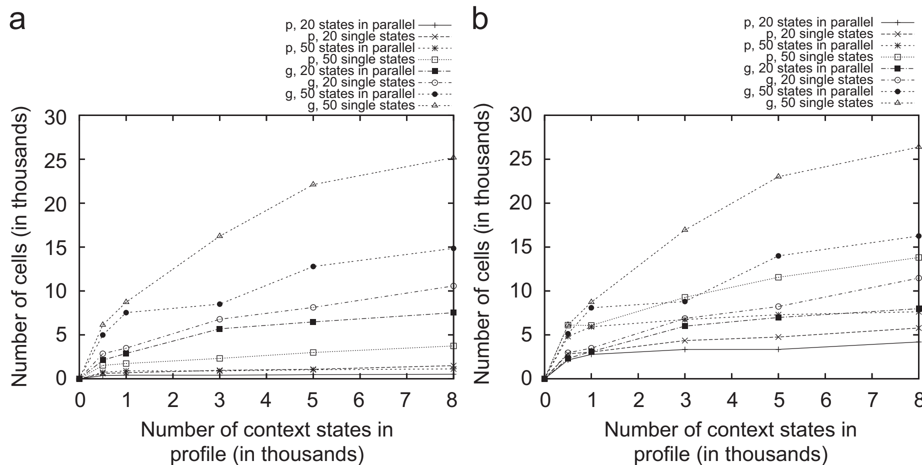


Fig. 15. Cell accesses for finding preferences related to queries for synthetic profiles using the query tree and the query graph for synthetic profiles for (a) exact match and (b) non exact match. With *p* we denote the profile tree and with *g* the preference graph.

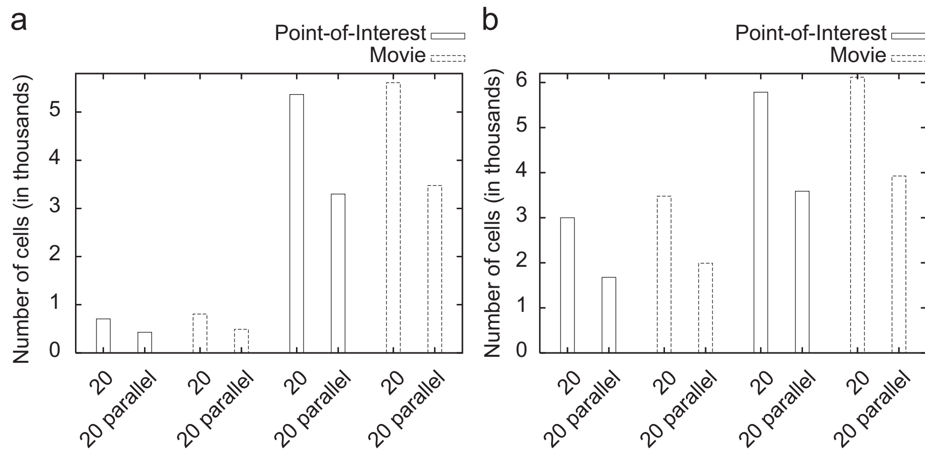


Fig. 16. Cell accesses for finding preferences related to queries using the query tree and the query graph for real profiles for (a) exact match and (b) non-exact match.

two different approaches for expressing preferences: a quantitative and a qualitative one. In the *quantitative* approach (e.g., [3,4,18]), preferences are expressed indirectly by using scoring functions that associate a numeric score with every tuple of the query answer. In this work, we have adapted the general quantitative framework of Agrawal and Wimmers [3], since it is easier for users to employ than the qualitative one. In the quantitative framework of Koutrika and Ioannidis [5,14], user preferences are stored as degrees of interest in *atomic query elements* (such as individual selection or join conditions) instead of interests in specific attribute values. Our approach can be generalized for this framework as well, by making the degree of interest for each atomic query element depends on context. In the *qualitative* approach (for example, [1,2,19]), the preferences between tuples in the answer to a query are specified directly, typically using binary preference relations. This framework can also be readily extended to include context.

There has been much work on developing a variety of context infrastructures and context-aware middleware and applications (such as the Context Toolkit [20] and the Dartmouth Solar System [21]). However, although there is much research on location-aware query processing in the area of spatio-temporal databases, integrating other forms of context in query processing is less explored. In the context-aware query processing framework of Feng et al. [22], there is no notion of preferences, instead context parameters are treated as normal attributes of relations. Recently, context has been used in information filtering to define context-aware filters which are filters that have attributes whose values change frequently [23].

Storing context data using data cubes, called context cubes, is proposed in [24] for developing context-aware applications that use archive sensor data. In this work, data cubes are used to store historical context data and to extract interesting knowledge from large collections of context data. The Context Relational Model (CR) introduced in [25] is an extended relational model that allows attributes to exist under some contexts or to have different values under different contexts. CR treats context as a first-class citizen at the level of data models, whereas in our approach, we use the traditional relational model to capture context as well as context-dependent preferences. Context as a set of dimensions (e.g., context parameters) is also considered in [26] where the problem of representing context-dependent semistructured data is studied, while in [27], an overview of a Multidimensional Query Language is given, that may be used to express context-driven queries. A context model is also deployed in [28] for enhancing web service discovery with contextual parameters. In [29], the current contextual state of a system is represented as a multidimensional subspace within or near other situation subspaces.

Extending the typical recommendation systems beyond the two dimensions of users and items to include further contextual information is studied in [30]. Contextual information is modeled using a number of parameters with hierarchical structure [31,30]. Using context is shown to improve the prediction of customer behavior.

There has been some recent work on contextual preferences. In [32], authors consider ranking database results based on contextual qualitative preferences. Context parameters are part of the database schema, while in our approach, context parameters are considered to be outside the database. Furthermore, our context parameters have a hierarchical nature that we explore in context resolution. A knowledge-based context-aware query preference model is proposed in [33], where context parameters are treated as normal attributes of relations. Contextual preferences, called situated preferences, are also discussed in [34]. In this approach, a context state is represented as a situation. Situations are uniquely linked through an N:M relationship with preferences expressed using the qualitative approach. Again, the context model is not hierarchical.

Finally, note that a preliminary abridged version of this paper appears in [35]. The preference graph, computing scores, multi-state resolution, various other enhancements and most of the experiments are new here.

In other previous work [36], we have addressed the same problem of expressing contextual preferences. However, the model used there for defining preferences includes only a single context parameter. Interest scores of preferences involving more than one context parameter are computed by a simple weighted sum of the preferences expressed by single context parameters. Here, we extend context descriptors, so that contextual preferences involve more than one context parameter and also, associate context with queries. Context resolution is also new here. In [38], we focus on the efficient execution of contextual queries. In particular, we are interested in creating groups of similar preferences for which we precompute rankings of database tuples.

8. Conclusions

The focus of this paper is on annotating database 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. Database preferences are augmented with context descriptors that specify the context states under which a preference holds. Similarly, each query is related with a set of context states. We consider the problem of identifying those preferences whose context states (as specified by their context descriptors) are the most similar to that of a given query. We call this problem *context resolution*. To realize context resolution, we propose two data structures, namely the *preference graph* and the *profile tree*, which allow for a compact representation of the context-dependent preferences.

To evaluate the usefulness of our model, we have performed two usability studies. Our studies have showed that annotating preferences with context improves the quality of the retrieved results considerably. The burden of having to specify contextual preferences is reasonable and can be reduced by providing users with default preferences that they can edit. We have also performed a set of experiments to evaluate the performance of context resolution using both real and synthetic datasets. The proposed data structures were shown to improve both the storage and the processing overheads. In general, the profile tree is more space-efficient than the preference graph. It also clearly outperforms the preference graph in the case of exact matches. The main advantage of the preference graph is the possibility for an incremental refinement of a context state. In particular, at each step of the resolution algorithm, we get a state that is closer to that of the query. This is not possible with the profile tree.

There are many directions for future work. One is to extend our model so as to support non-strict hierarchies. Although the cover relation is still valid in this case, this will require a revision of our definitions of distances between context states and possibly small modifications of the proposed data structures. Another direction for future work is preference application. In our current work, we assume that preferences are applied after the execution of a query to rank its result. Re-writing the query to incorporate contextual preferences is a promising alternative.

References

[1] J. Chomicki, Preference formulas in relational queries, ACM Trans. Database Syst. 28 (4) (2003) 427–466 ISSN: 0362-5915.
[2] W. Kießling, Foundations of preferences in database systems, in: VLDB, 2002, pp. 311–322.
[3] R. Agrawal, E.L. Wimmers, A framework for expressing and combining preferences, SIGMOD Rec. 29 (2) (2000) 297–306 ISSN 0163-5808.
[4] V. Hristidis, N. Koudas, Y. Papakonstantinou, PREFER: a system for the efficient execution of multi-parametric ranked queries, in: SIGMOD, 2001, pp. 259–270.
[5] G. Koutrika, Y. Ioannidis, Personalized queries under a generalized preference model, in: ICDE, 2005, pp. 841–852.
[6] A.K. Dey, Understanding and using context, Personal Ubiquitous Comput. 5 (1) (2001) 4–7 ISSN 1617-4909.
[7] M. Bazire, P. Brézillon, Understanding context before using It, in: CONTEXT, 2005, pp. 29–40.
[8] G. Chen, D. Kotz, A survey of context-aware mobile computing research, Technical Report TR2000-381, Dartmouth College, Computer Science ⟨ftp://ftp.cs.dartmouth.edu/TR/TR2000-381.ps.Z⟩, 2000.
[9] C. Bolchini, C. Curino, E. Quintarelli, F.A. Schreiber, L. Tanca, A data-oriented survey of context models, SIGMOD Rec. 36 (4) (2007) 19–26.
[10] B. Mobasher, R. Cooley, J. Srivastava, Automatic personalization based on Web usage mining, Commun. ACM 43 (8) (2000) 142–151.
[11] M. Ester, J. Kohlhammer, H.-P. Kriegel, The DC-tree: a fully dynamic index structure for data warehouses, in: ICDE, 2000, pp. 379–388.
[12] P. Vassiliadis, S. Skiadopoulos, Modelling and optimisation issues for multidimensional databases, in: CAiSE, 2000, pp. 482–497.
[13] G.A. Miller, WordNet: a lexical database for English, Commun. ACM 38 (11) (1995) 39–41 ISSN 0001-0782.
[14] G. Koutrika, Y.E. Ioannidis, Constrained optimalities in query personalization, in: SIGMOD, 2005, pp. 73–84.
[15] W. Kießling, G. Köstler, Preference SQL – design, implementation, experiences, in: VLDB, 2002, pp. 990–1001.
[16] Stanford Movie Database URL ⟨http://kdd.ics.uci.edu/databases/movies/movies.html⟩.
[17] C. Buckley, E.M. Voorhees, Retrieval evaluation with incomplete information, in: SIGIR, 2004, pp. 25–32.
[18] C. Li, K.C.-C. Chang, I.F. Ilyas, S. Song, RankSQL: Query algebra and optimization for relational top-k queries, in: SIGMOD, 2005, pp. 131–142.
[19] P. Georgiadis, I. Kapantaidakis, V. Christophides, E.M. Nguer, N. Spyratos, Efficient rewriting algorithms for preference queries, in: ICDE, 2008, pp. 1101–1110.
[20] D. Salber, A.K. Dey, G.D. Abowd, The context toolkit: aiding the development of context-enabled applications, in: CHI, 1999, pp. 434–441.
[21] G. Chen, M. Li, D. Kotz, Design and implementation of a largescale context fusion network, in: MobiQuitous, 2004, pp. 246–255.
[22] L. Feng, P.M.G. Apers, W. Jonker, Towards context-aware data management for ambient intelligence, in: DEXA, 2004, pp. 422–431.
[23] J.-P. Dittrich, P.M. Fischer, D. Kossmann, AGILE: adaptive indexing for context-aware information filters, in: SIGMOD, 2005, pp. 215–226.
[24] L.D. Harvel, L. Liu, G.D. Abowd, Y.-X. Lim, C. Scheibe, C. Chatham, Context cube: flexible and effective manipulation of sensed context data, in: Pervasive, 2004, pp. 51–68.
[25] Y. Roussos, Y. Stavrakas, V. Pavlaki, Towards a context-aware relational model, in: CRR, 2005, pp. 5–8.
[26] Y. Stavrakas, M. Gergatsoulis, Multidimensional semistructured data: representing context-dependent information on the web, in: CAiSE, 2002, pp. 183–199.
[27] Y. Stavrakas, K. Pristouris, A. Efandis, T.K. Sellis, Implementing a query language for context-dependent semistructured data, in: ADBIS, 2004, pp. 173–188.
[28] C. Doulkeridis, M. Vazirgiannis, Querying and updating a context-aware service directory in mobile environments, Web Intell. (2004) 562–565.
[29] A. Padovitz, S.W. Loke, A. Zaslavsky, Towards a Theory of Context Spaces, PerCom 00 (2004) 38.
[30] G. Adomavicius, R. Sankaranarayanan, S. Sen, A. Tuzhilin, Incorporating contextual information in recommender systems using a multidimensional approach, ACM Trans. Inf. Syst. 23 (1) (2005) 103–145.

[31] C. Palmisano, A. Tuzhilin, M. Gorgoglione, Using context to improve predictive modeling of customers in personalization applications, IEEE Trans. Knowl. Data Eng. 20 (11) (2008) 1535–1549.
[32] R. Agrawal, R. Rantzau, E. Terzi, Context-sensitive ranking, in: SIGMOD, 2006, pp. 383–394.
[33] A.H. van Bunningen, L. Feng, P.M.G. Apers, A context-aware preference model for database querying in an ambient intelligent environment, in: DEXA, 2006, pp. 33–43.
[34] S. Holland, W. Kießling, Situated preferences and preference repositories for personalized database applications, in: ER, 2004, pp. 511–523.
[35] K. Stefanidis, E. Pitoura, P. Vassiliadis, Adding context to preferences, in: ICDE, 2007, pp. 846–855.
[36] K. Stefanidis, E. Pitoura, P. Vassiliadis, A context-aware preference database system, Int. J. Pervasive Comput. Commun. 3 (4) (2007) 439–460.
[37] P. Brézillon, Context in artificial intelligence: I. A survey of the literature, Comput. Artif. Intell. 18 (4) (1999).
[38] K. Stefanidis, E. Pitoura, Fast contextual preference scoring of database tuples, in: EDBT, 2008, pp. 344–355.
[39] C. Bolchini, C. Curino, G. Orsi, E. Quintarelli, R. Rossato, F. Schreiber, L. Tanca, And what can context do for data? Commun ACM 52 (11) (2009) 136–140.
[40] T. Strang, C. Linnhoff-Popien, A context modeling survey, in: Workshop on Advanced Context Modelling, Reasoning and Management, 2004.