On Relaxing Contextual Preference Queries

Kostas Stefanidis, Evaggelia Pitoura and Panos Vassiliadis
Computer Science Department, University of Ioannina, GR-45110 Ioannina, Greece
{kstef, pitoura, pvassil}@cs.uoi.gr

Abstract

Personalization systems exploit preferences for providing users with only relevant data from the huge volume of information that is currently available. We consider preferences that depend on context, such as the location of the user. We model context as a set of attributes, each taking values from hierarchical domains. Often, the context of the query may be too specific to match any of the given preferences. In this paper, we consider possible expansions of the query context produced by relaxing one or more of its context attributes. A hierarchical attribute may be relaxed upwards by replacing its value by a more general one, downwards by replacing its value by a set of more specific values or sideways by replacing its value by sibling values in the hierarchy. We present an algorithm based on a prefix-based representation of context for identifying the preferences whose context matches the relaxed context of the query and some initial performance results.

1. Introduction

Personalization systems aim at providing users with only the data that is of interest to them from the huge amount of available information. One way to achieve personalization is through preferences [3, 5]. In our previous work [7, 8], we have argued for an extended preference model, where preferences depend on context. We have considered an extended preference model, where preferences depend on context. Context is a general term used to capture conditions such as time or location. We model context as a multidimensional entity, where each dimension corresponds to one context parameter. In the case of $n$ context parameters, a context state is an $n$-tuple with one value from its domain assigned to each of the $n$ context parameters. To enhance the expressiveness of our context model, context parameters take values from hierarchical domains. For instance, a context parameter location may take values from a neighborhood, city or country domain.

Each query is associated with one or more context states. Often the context of a query is too specific to match any of the available preferences. To handle this issue, we consider hierarchical relaxation as the approach of replacing the value of one context attribute by a corresponding value at a different level of abstraction. Various related context states may be produced by relaxing one or more of the context attributes. Such relaxations are ranked according to their similarity to the original query state. Similarity is defined based on the number of attributes that are relaxed and the associated depth of such relaxations.

We present an algorithm for identifying the preferences whose context states match best the relaxed context of a query. The algorithm uses a prefix-based representation for the set of context states of both the query and the preferences. We also present initial performance results regarding the cost of relaxation and the size of the produced results.

2. A Contextual Preference Model

In this section, we present our multidimensional preference model. As a running example, we consider a simple database with information about points_of_interest such as museums, archaeological places or zoos. The database schema consists of a single database relation: Points_of_Interest(pid, name, type, address, open-air, hours_of_operation, admission_cost). Context is modeled through a finite set of special-purpose attributes, called context parameters ($C_i$). In our example, we consider three context parameters as relevant: location, weather and accompanying_people. Each context parameter $C_i$ is characterized by a context domain $\text{dom}(C_i)$, that is an infinitely countable set of values. To allow flexibility in defining preferences, we model context parameters as multidimensional attributes. In particular, we assume that each context parameter participates in an associated hierarchy of levels of aggregated data [8]. We use the notation $\text{dom}_{L_j}(C_i)$ for the domain of level $L_j$ of parameter $C_i$. Fig. 1 depicts the hierarchies used in our example. Note that, we require that all values are grouped into the single value ‘all’ that corresponds to the top level $\text{ALL}$.

For a context value $c_i$, we use the notation $\text{level}(c_i)$ to refer to the level $L_j$, such that, $c_i \in \text{dom}_{L_j}(C_i)$. The relationship between the values of the context levels is
achieved through the use of a set of $\preceding C_i^n$, $L_i \prec L_j$, functions. A function $\preceding C_i^n$ assigns a value of the domain of $L_i$ to a value of the domain of $L_j$. For instance, $\preceding \text{Country} (\text{Athens}) = \text{Greece}$. The function $\text{desc} C_i^n$ is the inverse of $\preceding C_i^n$. We define the extended domain for a parameter $C_i$ with $m$ levels as $\text{edom}(C_i) = \bigcup_{i=1}^m \text{dom}(C_i)$. An context state $s$ is a $n$-tuple of the form $(c_1, c_2, \ldots, c_n)$, where $c_i \in \text{edom}(C_i)$. For example, a context state may be $(\text{Athens, warm, friends})$.

Users express conditions regarding the values of a single context parameter through context parameter descriptors.

**Definition 1 (Context parameter descriptor)** A context parameter descriptor $\text{cod}(C_i)$ for a parameter $C_i$ is an expression of the form: $C_i \in \{\text{value}_1, \ldots, \text{value}_m\}$, where $\text{value}_k \in \text{edom}(C_i)$, $1 \leq k \leq m$.

An example context parameter descriptor for location is $\text{location} \in \{\text{Plaka, Ioannina}\}$. Given a set of context parameters $C_1, \ldots, C_n$, a composite context descriptor describes a set of context states, with each state having a specific value for each parameter.

**Definition 2 (Composite context descriptor)** A composite context descriptor $\text{cod}(C_i)$ is a formula $\text{cod}(C_i) \wedge \text{cod}(C_j) \wedge \ldots \wedge \text{cod}(C_k)$ where each $C_i$, $1 \leq j \leq k$ is a context parameter and there is at most one parameter descriptor per context parameter $C_i$.

For instance, the context descriptor $(\text{location} \in \{\text{Plaka}\} \wedge \text{weather} \in \{\text{warm, hot}\} \wedge \text{accompanying people} \in \{\text{friends}\})$ corresponds to the following two context states: $(\text{Plaka, warm, friends})$ and $(\text{Plaka, hot, friends})$. If a context descriptor does not contain a context parameter $C_i$, this means that the absent context parameter is irrelevant. This is equivalent to a condition $C_i \in \{\text{all}\}$.

To achieve personalization, users express their preference for specific database instances by providing a numeric score between 0 and 1. Value 1 indicates extreme interest, while value 0 indicates lack of interest. Interest is expressed for specific values of attributes of the database relations. In particular:

**Definition 3 (Contextual preference)** A contextual preference is a triple of the form $(\text{cod}, \text{attr}\_\text{clause}, \text{int}\_\text{score})$, where $\text{cod}$ is a context descriptor, $\text{attr}\_\text{clause}$ is a selection condition including attributes of a relation and $\text{int}\_\text{score}$ is a real number between 0 and 1.

The meaning is that in the set of context states specified by $\text{cod}$, all database tuples for which the $\text{attr}\_\text{clause}$ holds have the indicated interest score. For example, consider that a user wants to express the fact that, when she is in Plaka and the weather is warm, she likes to visit Acropolis. This may be expressed through the contextual preference: $((\text{location} \in \{\text{Plaka}\} \wedge \text{temperature} \in \{\text{warm}\}$, (name = Acropolis), 0.9).

A profile $P$ is a set of contextual preferences.

### 3. Hierarchical Context Relaxation

Each query is enhanced with information regarding context. We assume that the context of a query is also expressed through a composite context descriptor. Usually, the context associated with a query is the context surrounding the user at the time of the submission of the query. Besides this implicit context, queries may also be explicitly augmented with a context descriptor to allow exploratory queries, for instance: “What places should I visit during my family vacations in Athens this summer (implying good weather)?”. Let $P_C$ be the set of context states that correspond to a given profile $P$ and $Q_C$ be the set of context states that correspond to a query $Q$. We are interested in computing the set $P_C \cap Q_C$ and returning any related preferences.

It is possible that the set $P_C \cap Q_C$ is empty or that the preferences associated with its elements are not enough for achieving an effective personalization of the query. Further, it may not be possible to express the context of a query precisely. Thus, we consider ways of relaxing the query context. First, we focus on relaxing context parameter descriptors. To achieve this, we introduce relaxation operators. A hierarchical context parameter may be relaxed upwards by replacing its value with a more general value or downwards by replacing its value by a set of more specific values. To quantify the degree of hierarchical relaxation, we define first the distance between two levels as follows:

**Definition 4 (Level distance)** Given two levels $L_i$ and $L_j$, their distance $d_H(L_i, L_j)$ is defined as follows:

1. if a path exists in a hierarchy between $L_i$ and $L_j$, then $d_H(L_i, L_j)$ is the minimum number of edges that connect $L_i$ and $L_j$,
2. otherwise, $d_H(L_i, L_j) = \infty$.

A context parameter descriptor can be relaxed upwards by allowing a parameter to take a more general value, that is, a value at a higher level of the hierarchy.
Definition 5 (Upwards relaxed CoD) Given a context parameter descriptor $\text{cod}(C_i)$: $C_i \in S$, for a parameter $C_i$, $\text{up}(\text{cod}(C_i), r)$ is the expression $C_i \in S'$, where $S' = \{ v' \mid v' = \text{anc}_{\text{level}(v)}(\text{level}(v')), v \in S$ and $d_H(\text{level}(v), \text{level}(v')) = r \}$.

We call the parameter $r$ relaxation depth. For instance, $\text{up}(\text{location} \in \{\text{Plaka}\}, 1)$ is $\{\text{Athens}\}$, while $\text{up}(\text{location} \in \{\text{Plaka}\}, 2)$ is $\{\text{Greece}\}$. Similarly, a context parameter descriptor can also be relaxed downwards:

Definition 6 (Downwards relaxed CoD) Given a context parameter descriptor $\text{cod}(C_i)$: $C_i \in S$, for a parameter $C_i$, $\text{down}(\text{cod}(C_i), r)$ is the expression $C_i \in S'$, where $S' = \{ v' \mid v' = \text{desc}_{\text{level}(v)}(\text{level}(v')), v \in S$ and $d_H(\text{level}(v), \text{level}(v')) = r \}$.

For example, $\text{down}(\text{weather} \in \{\text{good}\}, 1)$ is the set of values $\{\text{mild}, \text{warm}, \text{hot}\}$. Finally, a context parameter may be relaxed sideways by being replaced by one or more values that belong to the same level of the hierarchy. To quantify the depth of such sideways relaxation, we use the following definition of sibling value distance:

Definition 7 (Sibling value distance) The sibling value distance of two context values $c_1$ and $c_2 \in \text{dom}(C_i)$, with $\text{level}(c_1) = \text{level}(c_2)$ is defined as follows:

$$d_S(c_1, c_2) = |\text{level}(\text{lca}(c_1, c_2)) - \text{level}(c_1)|,$$

where $\text{lca}(c_1, c_2)$ is the least common ancestor of $c_1$ and $c_2$.

Definition 8 (Sideways relaxed CoD) Given a context parameter descriptor $\text{cod}(C_i)$: $C_i \in S$, for a parameter $C_i$, $\text{side}(\text{cod}(C_i), r)$ is the expression $C_i \in S'$, where $S' = \{ v' \mid \text{level}(v') = \text{level}(v), u \in S$ and $d_S(v, v') = r \}$.

For example, $\text{side}(\text{weather} \in \{\text{hot}\}, 1)$ is the set of values $\{\text{mild}, \text{warm}, \text{hot}\}$. Note that using the sibling distance, the distance of two values at the same level depends on how far up the hierarchy their first common ancestor is located. For example, $d_S(\text{hot}, \text{warn}) = 1$, while $d_S(\text{hot}, \text{cold}) = 2$.

Having defined the upwards, downwards and sideways relaxed CoD, we define the overall distance between two context parameters descriptors, as the minimum possible distance for all directions for all members of a CoD.

Definition 9 (Overall CoD distance) Given two context parameter descriptors $\text{cod}^1(C_i)$: $C_i \in S_1$, and $\text{cod}^2(C_i)$: $C_i \in S_2$, the distance between the two context parameter descriptors is the minimum distance $r$ such that one of the following holds:

- $\text{cod}^2(C_i) \cap \text{up}(\text{cod}^1(C_i), r) \neq \emptyset$
- $\text{cod}^2(C_i) \cap \text{down}(\text{cod}^1(C_i), r) \neq \emptyset$
- $\text{cod}^2(C_i) \cap \text{side}(\text{cod}^1(C_i), r) \neq \emptyset$

Assume a query $Q$ with a context descriptor $\text{cod}^Q = \text{cod}(C_1) \land \text{cod}(C_2) \ldots \land \text{cod}(C_n)$, where any missing context parameter descriptor $C_i$ is replaced by the descriptor $C_i \in \{\text{all}\}$. We can relax $Q$ by relaxing upwards, downwards or sideways any subset of the $n$ context parameter descriptors. Next, we define the distance between the original $\text{cod}^Q$ and a relaxed context descriptor that results by relaxing one or more of its constituting context parameter descriptors.

Definition 10 (Distance between composite CoDs) Assume two context descriptors $\text{cod}_1 = \text{cod}^1(C_1) \land \text{cod}^1(C_2) \ldots \land \text{cod}^1(C_n)$ and $\text{cod}_2 = \text{cod}^2(C_1) \land \text{cod}^2(C_2) \ldots \land \text{cod}^2(C_n)$. Then, the distance between the two composite context descriptors is the sum of the individual distances of context parameter descriptors:

$$\text{dist}(\text{cod}_1, \text{cod}_2) = \sum_{i=1}^{n} [\text{dist}(\text{cod}^1(C_i), \text{cod}^2(C_i))].$$

To compute the distance between a context descriptor and a state, we must simply transform the state to a composite context descriptor.

Definition 11 (Distance between state and composite CoD) Assume a context descriptor $\text{cod}_1 = \text{cod}^1(C_1) \land \text{cod}^1(C_2) \ldots \land \text{cod}^1(C_n)$, and, a state $s^2 = (c_1^2, c_2^2, \ldots, c_n^2)$. Construct a context descriptor $\text{cod}_2 = \text{cod}^2(C_1) \land \text{cod}^2(C_2) \ldots \land \text{cod}^2(C_n)$, s.t., $\text{cod}^2(C_i) = c_i^2$. Then, the distance between $\text{cod}_1$ and $s^2$ is

$$\text{dist}(\text{cod}_1, s^2) = \text{dist}(\text{cod}_1, \text{cod}_2).$$

Finally, to construct the distance between two states, we simply need to construct the appropriate descriptors and measure their distance.

Definition 12 (Distance between states) Assume two states $s^1 = (c_1^1, c_2^1, \ldots, c_n^1)$ and $s^2 = (c_1^2, c_2^2, \ldots, c_n^2)$. Construct two context descriptor $\text{cod}_1 = \text{cod}^1(C_1) \land \text{cod}^1(C_2) \ldots \land \text{cod}^1(C_n)$, s.t., $\text{cod}^1(C_i) = c_i^1$, $i \in \{1, 2\}$, $j \in \{1, \ldots, n\}$. Then, the distance between $s^1$ and $s^2$ is

$$\text{dist}(s^1, s^2) = \text{dist}(\text{cod}_1, \text{cod}_2).$$

Often, the relaxation of a state $s$ may result in two states $s^1$ and $s^2$ that are equally similar to $s$, that is, $\text{dist}(s^1, s) = \text{dist}(s^2, s)$. In this case, we use the Jaccard distance between the two states $s^1$ and $s^2$ to select one of them. Intuitively, this allows us to select the “largest” state in terms of the cardinality, in particular, we choose the state that results in the largest number of values at level $L_1$ (the lowest level of detail). The motivation is that (in the absence of any other information), this is the state that is more likely to appear. The formal definitions can be found in [9].

4. A Relaxed Context Resolution Algorithm

The question that arises is which of the context parameters of the query context to relax and how much (i.e., what
is an appropriate relaxation depth, \( r \) so that a large enough set of preferences in \( P_C \) match the context state of the relaxed query. We call this problem relaxed context resolution. We assume that the system (or the user) associates a value \( k \) with each query that specifies how many matching preferences from the profile should be returned. In particular, given a query \( Q \) with a context descriptor \( cod^Q \), we look for \( k \) preferences in \( P_C \) that match the set of context states specified by \( cod^Q \). As long as there are less than \( k \) preferences, we relax a number of the context parameter descriptors in \( cod^Q \) by using gradually larger relaxation depths.

**Data Structures.** To store the contextual preferences in \( P \), we use a data structure called profile tree [8]. Let \( P_C \) be the set of context states of all context descriptors that appear in \( P \). The basic idea is to store in the profile tree the context states in \( P_C \) so that there is exactly one path in the tree for each context state \( s \in P_C \). Specifically, the profile tree for \( P_C \) is a directed acyclic graph with a single root node and \( n + 1 \) levels. Each one of the first \( n \) levels corresponds to a context parameter. For simplicity, assume that context parameter \( C_i \) is mapped to level \( i \). Each non-leaf node at level \( k \) maintains cells of the form \([key, pointer]\), where \( key \in edom(C_k) \) for some value of \( c_k \) that appeared in a state \( s \in P_C \). No two cells within the same node contain the same key value. The pointer points to a node at level \( k + 1 \) having cells with key values in \( edom(C_{k+1}) \) which appeared in the same state \( s \) with the key. Each leaf node maintains the part \([attr\_clause, int\_score]\) of the preference associated with the path (context state) leading to it.

Let \( Q_C \) be the set of context states derived from the descriptor \( cod^Q \) of query \( Q \). As opposed to [8], where we used the profile tree to check for each individual \( s \) in \( Q_C \), the proposed algorithm tests for all states in \( Q_C \) within a single pass of the profile tree. To achieve this, the context states in \( Q_C \) are also represented by a data structure similar to the profile tree, that we call the Query tree, so that there is exactly one path in the tree for each context state \( s \in Q_C \).

**A Context Resolution Algorithm.** The Relaxed Context Resolution Algorithm (Algorithm 1) gradually relaxes the query context, so that enough preferences are found. Initially, we start by searching for states in the profile tree that are the same with the states of the query. In a breadth-first manner, we search for pairs of nodes that belong to the same level. Each pair consists of a node of the query tree and a node of the profile tree. Initially, there is one pair of nodes, \((R_Q, R_P)\) (level \( i = 0 \)). For each value of the query node \( R_Q \) that is equal to a value of the profile node \( R_P \), we create a new pair of nodes \((R_Q \rightarrow child, R_P \rightarrow child)\). These nodes refer to the next level \( (i + 1) \). After having checked 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 \( n \), if a value of a query node is equal to a value of a profile node, we retrieve from its leaf node the attribute clause with its relative interest score. If the number of preferences returned are less than the desired number \( k \), we relax the query conditions in rounds. First, we look for relaxed states with distance equal to 1 from the searching context states, i.e., the states of the query tree. \( SN, SN' \) sets of pairs of nodes, each pair related with a distance value. Initially: \( SN = \{(R_Q, R_P, 0)\}, SN' = \{\} \)

**Begin**

\[
\text{distance} = 0
\]

while ResultSet less than \( k \) do

for level \( i = 0 \) to \( n - 1 \) do

for each pair \( sn \in SN \), with \( sn = (q\_node, p\_node, dist) \) do

\[
\forall x \in q\_node
\]

\[
\forall y \in p\_node
\]

if \( \text{dist} + d(x, y) \leq \text{distance} \) then

\[
SN' = SN' \cup \{(x \rightarrow \text{child}, y \rightarrow \text{child}, \text{dist} + d(x, y))\}
\]

else if \( i = n - 1 \) then

\[
\text{attr}\_\text{clause} = y \rightarrow \text{child} \text{attr}\_\text{clause}
\]

\[
\text{int}\_\text{score} = y \rightarrow \text{child} \text{int}\_\text{score}
\]

ResultSet = ResultSet \cup (\text{attr}\_\text{clause}, \text{int}\_\text{score})

end if

end for

end if

\[
SN = SN'
\]

\[
SN' = \{\}
\]

end for

\[
\text{distance}++
\]

end while

End

**5. Performance Evaluation**

In this section, we present initial performance results regarding the relaxation algorithm. There are three context parameters, each one having a domain with 100 values. Profiles have various numbers of context states (from 500 to 8000). The values of two of the context parameters are drawn from their corresponding domain using a zipf data distribution with \( \alpha = 2.0 \), while the values of the third parameter are selected using a uniform data distribution.

First, we report on a number of experiments that compare the performance of searching for matching states for
Contextual preferences, called situated preferences, are introduced in [1]. Situations (i.e., context states) are uniquely linked through an N:M relationship with preferences. A knowledge-based context-aware query preference model is also proposed in [10]. In our previous research [7, 8], we have addressed the problem of expressing contextual preferences. Here we extend this work to allow relaxing the specification of the query context. For measuring the distance between two context values, we just use their path distance in the hierarchy and their Jaccard distance to resolve ties. There has been a lot of work on defining the semantic similarity of words connected through a lexical hierarchy (e.g., [6]) that we plan to explore in future work. Query relaxation has attracted some attention recently. A framework to relax queries involving numeric conditions in selection and join predicates is proposed in [4]. In this paper, we focus on categorical attributes with hierarchical domains. The relaxation algorithm proposed in [2] produces a relaxed query for a given initial range query and a desired cardinality of the result set. Again, this work considers numerical attributes.

6. Related Work

Contextual preferences, called situated preferences, are introduced in [1]. Situations (i.e., context states) are uniquely linked through an N:M relationship with preferences. A knowledge-based context-aware query preference model is also proposed in [10]. In our previous research [7, 8], we have addressed the problem of expressing contextual preferences. Here we extend this work to allow relaxing the specification of the query context. For measuring the distance between two context values, we just use their path distance in the hierarchy and their Jaccard distance to resolve ties. There has been a lot of work on defining the semantic similarity of words connected through a lexical hierarchy (e.g., [6]) that we plan to explore in future work. Query relaxation has attracted some attention recently. A framework to relax queries involving numeric conditions in selection and join predicates is proposed in [4]. In this paper, we focus on categorical attributes with hierarchical domains. The relaxation algorithm proposed in [2] produces a relaxed query for a given initial range query and a desired cardinality of the result set. Again, this work considers numerical attributes.

7. Summary

In this paper, we consider context-dependent preferences, which are preferences that depend on context. Context is modeled as a multidimensional entity with each of its dimensions taking values from a hierarchical domain. Each query is also augmented with context information. We focus on the problem of relaxing the context of the query, so that there are enough preferences whose associated context match that of the query. We consider relaxing a hierarchical value by using a more general (upwards relaxation), a more specific (downwards relaxation) or a sibling (sideways relaxation) one. Depending on the distance in the associated hierarchy between the original and the relaxed value, we define different relaxation levels. We also present an algorithm that incrementally relaxes the query context, until a sufficiently large number of results is produced.

Acknowledgment

This paper is part of the 03-ED-591 research project, implemented within the framework of the "Reinforcement
Programme of Human Research Manpower” (PENED) and co-financed by National and Community Funds (25% from the Greek Ministry of Development-General Secretariat of Research and Technology and 75% from E.U.-European Social Fund).

References


