ΤΕΧΝΙΚΗ ΑΝΑΦΟΡΑ • TECHNICAL REPORT

On Relaxing Contextual Preference Queries

K. Stefanidis, E. Pitoura and P. Vassiliadis

TECHNICAL REPORT
DCS 2007-01

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ • ΠΑΝΕΠΙΣΤΗΜΙΟ ΙΩΑΝΝΙΝΩΝ
DEPARTMENT OF COMPUTER SCIENCE • UNIVERSITY OF IOANNINA
http://www.cs.uoi.gr

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 out of the huge volume of the information that is currently available. Such preferences may dependent on context. We model context as a set of hierarchical attributes, each taking values from hierarchical domains. In this paper, we consider relaxing the context associated with a query, so that there are enough preferences with matching context. 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 consider possible expansions of the query context produced by relaxing one or more of its attributes in any of the above ways. We also present an algorithm based on a prefix-based representation of context for computing the preferences whose context matches best the relaxed context of the query.

1 Introduction

Personalization systems aim at providing users with only the data that is of interest to them out of the huge amount of available information. One way to achieve personalization is through preferences [5, 7]. With preferences, users express their degree of interest on specific pieces of information. In our previous work [9, 10], we have argued for 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. Then, in the case of n context parameters, a context state is a n-tuple with one value from its domain assigned to each of the n context parameters. To allow more flexibility in expressing context, we allow context parameters to take values from hierarchical domains. For instance, a context parameter *location* may take values from a *region*, *city* or *country* domain. Users employ context descriptors to express preferences on specific database instances for a variety of context states.

Each query is associated with one or more context state. 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. The general goal is to be able to rank the results of a query differently based on the preferences associated with a given query context.

Most 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 hierarchical context attribute by a corresponding value at a different level of abstraction. Hierarchical attribute relaxation may be *upwards*, in which case the value of an attribute is replaced by a more general value in the associated hierarchy, or *downwards*. in which case the value is replaced by a set of more specific values. We also consider *sideways* relaxation, by replacing a value of a context attribute by one or more of its sibling values in the hierarchy. All three relaxation types may differ in depth, where depth intuitively expresses how far away in the hierarchy is the initial value from the relaxed one. For instance, relaxing *Athens* to *Greece* has less depth than relaxing *Athens* to *Europe* and thus considered closer to it.

Various related context states may be produced by relaxing one or more of its attributes upwards, downwards or sideways. 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 computing 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.

The rest of this paper is structured as follows. In Section 2, we present our context-dependent preference model. The problem of context relaxation is introduced in Section 3, while an algorithm for finding the context states that match the relaxed context of a query is provided in Section 4. In Section 5, we present some initial performance results regarding the spread of the achieved relaxation. Section 6 includes a discussion of related work, while Section 7 concludes the paper.

2 The Multidimensional 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, monuments, archaeological places or zoos. The database schema consists of a single database relation: *Points_of_Interest(pid, name, type, location, open-air, hours_of_operation, admission_cost)*. We consider three context parameters as relevant: *location*, *weather* and *accompanying_people*. Depending on context, users prefer *points_of_interest* that have specific attribute values. For example, a point-of-interest of *type zoo* may be a preferred place to visit than a point-of-interest of *type brewery* in the context of *family*.

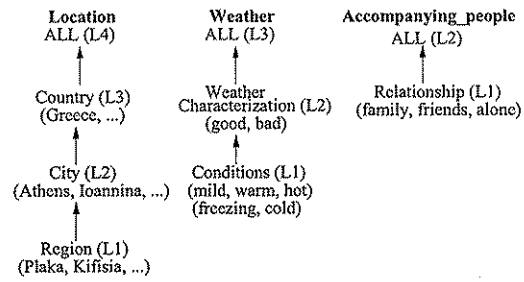

Figure 1: Example Hierarchies.

2.1 Modeling Context

Context is modeled through a finite set of special-purpose attributes, called *context parameters* (C_i). For a given application X, we define its context environment CE_X as a set of n context parameters $\{C_1, C_2, \ldots, C_n\}$. For instance, the context environment of our example is $\{location,\ weather,\ accompanying\_people\}$. Each context parameter C_i is characterized by a *context domain* $dom(C_i)$, that is an infinitely countable set of values. A *context state* w is a n–tuple of the form $(c_1, c_2, \ldots, c_n)$, where $c_i \in dom(C_i)$. For example, a context state may be: $(Plaka, warm, friends)$.

To allow more 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, i.e., it can be viewed from different levels of detail. Formally, an *attribute hierarchy* is a lattice $(L, \prec)$: $L = (L_1, \ldots, L_{m-1}, ALL)$ of m *levels* and $\prec$ is a partial order among the levels of L such that $L_1 \prec L_i \prec ALL$, for every $1 < i < m$. We require that the upper bound of the lattice is always the level ALL, so that we can group all values into the single value '*all*'. We use the notation $dom_{L_j}(C_i)$ for the domain of level L_j of parameter C_i. Regarding our running example, Fig. 1 depicts the hierarchies that are used.

For a context value c_i, we use the notation $level(c_i)$ to refer to the level L_j, such that, $c_i \in dom_{L_j}(C_i)$. The relationship between the values of the context levels is achieved through the use of the set of $anc_{L_i}^{L_j}$, $L_i \preceq L_j$, functions [12]. A function $anc_{L_i}^{L_j}$ assigns a value of the domain of L_i to a value of the domain of L_j. For instance, $anc_{Region}^{City}(Plaka) = Athens$. The function $desc_{L_i}^{L_j}$ is the inverse of $anc_{L_i}^{L_j}$, that is $desc_{L_l}^{L_j}(v)=\{x \in dom_{L_i}(C_k) : anc_{L_i}^{L_j}(x) = v\}$.

We define the extended domain for a parameter C_i with m levels as $edom(C_i)$ $= \cup_{j=1}^{m}\ dom_{L_j}(C_i)$. An *(extended) context state* s is a n–tuple of the form $(c_1, c_2, \ldots, c_n)$, where $c_i \in edom(C_i)$. The set of all possible extended context states called *extended world*, EW, is the Cartesian product of the extended domains of the context attributes: $EW = edom(C_1) \times edom(C_2) \times \ldots \times edom(C_n)$.

Users can express conditions regarding the values of a context parameter

through *context parameter descriptors.*

Definition 1 (Context parameter descriptor) *A context parameter descriptor $cod(C_i)$ for a parameter C_i is an expression of the form: $C_i \in \{value_1, \ldots, value_m\}$, where $value_k \in edom(C_i)$, $1 \leq k \leq m$.*

For example, given the context parameter *location*, a context parameter descriptor can be of the form $location \in \{Plaka, Ioannina\}$. A context descriptor is a specification that a user can make for a set of context parameters, through the combination of simple parameter descriptors.

Definition 2 (Composite context descriptor) *A (composite) context descriptor cod is a formula $cod(C_{i_1}) \wedge cod(C_{i_2}) \wedge \ldots \wedge cod(C_{i_k})$ where each C_{i_j}, $1 \leq j \leq k$ is a context parameter and there is at most one parameter descriptor per context parameter C_{i_j}.*

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. Clearly, one context descriptor can produce more than one state. For instance, the context descriptor (location = $Plaka \wedge$ weather $\in \{warm, hot\} \wedge$ accompanying_people = $friends$) corresponds to the following two context states: $(Plaka, warm, friends)$ and $(Plaka, hot, friends)$. If a context descriptor does not contain all context parameters, that means that the absent context parameters have irrelevant values. This is equivalent to a condition $C_i = all$.

2.2 Contextual Preferences

To achieve context-aware personalization, users express their preference for specific database instances by providing a numeric score which is a real number between 0 and 1 that expresses their degree of interest. Value 1 indicates extreme interest, while value 0 indicates lack of interest. Interest is expressed for specific values of non context attributes of the database relations. In particular, a *contextual preference* is defined as follows.

Definition 3 (Contextual preference) *A contextual preference is a triple of the form cp = (cod, attr_clause, int_score), where cod is a context descriptor, the attr_clause $\{A_1\theta_1 a_1, A_2\theta_2 a_2, \ldots, A_k\theta_k a_k\}$ specifies a set of attributes $A_1, A_2, \ldots, A_k$ with their values $a_1, a_2, \ldots, a_k$ with $a_i \in dom(A_i)$, $\theta_i \in \{=, <, > , \leq, \geq, \neq\}$ and int_score is a real number between 0 and 1.*

The meaning is that in the set of context states specified by *cod*, all database tuples (instances) for which the attributes $A_1, A_2, \ldots, A_m$ have respectively values $a_1, a_2, \ldots, a_m$ are assigned the indicated interest score. A user can define non contextual preference queries, by using empty context descriptors which correspond to the $(all, all, \ldots, all)$ state. In our reference example, as non-context parameters, we use the attributes of the relation $Points\_of\_Interest$.

For example, consider that a user wants to express the fact that, when she is at *Plaka* and the weather is *warm*, she likes to visit *Acropolis*. This may be expressed through the following contextual preference: cp_1 = ((location = *Plaka* $\wedge$ temperature = *warm*), (name = *Acropolis*), 0.8).

A *profile* P is a set of contextual preferences. We define the *active domain* $adom(C_i)$ for a context parameter C_i as the set of context values $c_i \in dom(C_i)$ that appeared in at least one context descriptor in P.

3 Multidimensional Context Relaxation

In this section, we consider the problem of ranking the contextual preferences in a profile based on how well they match the context of a query.

3.1 Contextual Queries

A contextual query is a query enhanced with information regarding context. Implicitly, the context associated with a query is the current context, that is, the context surrounding the user at the time of the submission of the query. The current context should correspond to a single context state, where each of the context parameters takes a specific value from its most detailed domain. However, in some cases, it may not be possible to specify the current context using one specific value. For example, this may occur, when the values of some context parameters are provided by sensor devices with limited accuracy. In this case, a context parameter may take a single value from a higher level of the hierarchy or even more than one value. Besides the implicit context, we also consider queries that are explicitly augmented with a context descriptor. For example, a user may want to submit an exploratory query of the form: "When I visit Athens with friends this summer (implying good weather), what places should I visit?". Thus, in general, a contextual preference query Q is a query enhanced with a context descriptor denoted cod^Q.

Let P_C be the set of all context states that correspond to a given profile P. Let Q_C be the set of all context states that correspond to a contextual query Q. The *context resolution problem* refers to the problem of computing the set $P_C \cap Q_C$ and returning any related contextual 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. In this case, the query context needs to be "relaxed".

In the following, we consider ways of relaxing the context of a query by introducing relaxed context descriptors so that the matching preferences returned are sufficient.

3.2 Relaxed Context Parameter Descriptors

If there are not enough preferences matching the context of a query, we relax the context descriptor of the query. To achieve this, we introduce relaxation

operators on context parameter descriptors, that is, on descriptors involving a single context parameter. A hierarchical context parameter may be relaxed *upwards* in which case its value is replaced by a more general value in the associated hierarchy, that is, by values at higher levels. Further, a hierarchical parameter may be relaxed *downwards* in which case its value is replaced by a set of more specific values, that is, by values at lower levels. To quantify the degree of hierarchical relaxations, we define the distance between two levels as follows:

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

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

A context parameter descriptor can be relaxed upwards by allowing a parameter to take a more general value:

Definition 5 (Upwards relaxed CoD) *Given a context parameter descriptor $cod(C_i)$: $C_i \in S$, for a parameter C_i, $up(cod(C_i), r)$ is the expression $C_i \in S'$, where $S' = \{v' \mid v' = anc_{level(v)}^{level(v')}(v), v \in S$ and $dist_H(level(v), level(v')) = r\}$.*

For instance, up((location $\in \{Plaka\}$), 1) is $\{Athens\}$, while up((location $\in \{Plaka\}$), 2) is $\{Greece\}$. A context parameter descriptor can also be relaxed downwards by allowing a parameter to take a more specific value. The main difference is that downwards relaxation may produce sets of values that are large.

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

For example, down((weather $\in \{good\}$), 1) is the set of values $\{mild, warm, hot\}$. We call the parameter r in the definitions above *relaxation depth*. The relaxation depth r specifies how many levels up or down the hierarchy should a parameter be relaxed.

A context parameter may be also replaced by a sibling value. To quantify this, we use the following definition of sibling value distance.

Definition 7 (Least common ancestor) *Given two context values c_1 and $c_2 \in edom(C_i)$, their least common ancestor, $lca(c_1, c_2)$ is value $c_3 \in edom(C_i)$ such that $anc_{level(c_1)}^{level(c_3)}(c_1) = anc_{level(c_2)}^{level(c_3)}(c_2)$ and there is no value $c_4 \in edom(C_i)$, such that, $anc_{level(c_1)}^{level(c_4)}(c_1) = anc_{level(c_2)}^{level(c_4)}(c_2)$ and $level(c_3) > level(c_4)$.*

Next, we define the distance between two sibling context values.

Definition 8 (Sibling value distance) *The sibling value distance of two context values c_1 and $c_2 \in edom(C_i)$, with $level(c_1) = level(c_2)$, is defined as follows:*

$$dist_S(c_1, c_2) = |level(lca(c_1, c_2)) - level(c_1)|.$$

Now, sideways relaxation is defined as:

Definition 9 (Sideways relaxed CoD) *Given a context parameter descriptor $cod(C_i)$: $C_i \in S$, for a parameter C_i, $side(cod(C_i), r)$ is the expression $C_i \in S'$, where $S' = \{v' \mid level(v') = level(v), u \in S$ and $dist_S(v', v) = r\}$.*

For example, side($(weather \in \{hot\})$, 1) is the set of values $\{mild, warm, hot\}$. 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, $d_S(hot, warm) = 1$, while, $d_S(hot, cold) = 2$.

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

- $cod^2(C_i) \bigcap up(cod^1(C_i), r) \neq \emptyset$
- $cod^2(C_i) \bigcap down(cod^1(C_i), r) \neq \emptyset$
- $cod^2(C_i) \bigcap side(cod^1(C_i), r) \neq \emptyset$

Assume a query Q with a context descriptor $cod^Q = cod(C_1) \wedge cod(C_2) \ldots \wedge cod(C_n)$, where any missing context parameter descriptor C_l is replaced by the descriptor $C_l \in \{all\}$.

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 cod^Q and a relaxed context descriptor that results by relaxing one or more of its constituting context parameter descriptors.

3.3 Relaxed Contextual Preference Selection

In following, we define the distance of two composite context descriptors. Then, we will define the distance of a composite context descriptor and a context state. Then, the distance between two context states is straightforward.

Definition 11 (Distance between composite CoDs) *Assume two context descriptors* $cod_1 = cod^1(C_1) \wedge cod^1(C_2) \ldots \wedge cod^1(C_n)$ *and* $cod_2 = cod^2(C_1) \wedge cod^2(C_2) \ldots \wedge cod^2(C_n)$. *Then, the distance between the two composite context descriptors is the sum of the individual distances of context parameter descriptors:*

$$dist(cod_1, cod_2) = \sum_{i=1}^{n} |dist(cod^1(C_i), cod^2(C_i))|.$$

To compute the distance between a context descriptor and a state, we must simply transform the state to a composite context descriptor. Then, the distance is defined as in Definition 13.

Definition 12 (Distance between state and composite CoD) *Assume a context descriptor* $cod_1 = cod^1(C_1) \wedge cod^1(C_2) \ldots \wedge cod^1(C_n)$, *and, a state* $s^2 = (c_1^2, c_2^2, \ldots, c_n^2)$. *Construct a context descriptor* $cod_2 = cod^2(C_1) \wedge cod^2(C_2) \ldots \wedge cod^2(C_n)$, *s.t.,* $cod^2(C_i)$: $C_i = c_i^2$. *Then, the distance between* cod_1 *and* s^2 *is*

$$dist(cod_1, s_2) = dist(cod_1, cod_2)$$

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

Definition 13 (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* $cod_i = cod^i(C_1) \wedge cod^i(C_2) \ldots \wedge cod^i(C_n)$, *s.t.,* $cod^i(C_j)$: $C_j = c_j^2$, $i \epsilon \{1, 2\}$, $j \epsilon \{1, \ldots, n\}$. *Then, the distance between* s^1 *and* s^2 *is:*

$$dist(s^1, s^2) = dist(cod_1, cod_2)$$

3.4 Resolving Ties

To resolve ties between equally ranked states, we use the Jaccard distance function that expresses the distance between two states. In this case, we compute all the descendants of each value of a state. For two values of two 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 this two sets. In this case, we consider as a better match, the "largest" state in terms of cardinality. Next, we define the Jaccard distance of two values c_1 and $c_2 \in edom(C_i)$, when either the one is the ancestor of the other or they are siblings, and use it to define the Jaccard state distance:

Definition 14 (Jaccard distance) *The Jaccard distance of two context values* c_1 *and* $c_2 \in edom(C_i)$ *is defined as:*

1. $dist_J(c_1, c_2) = 1 - \frac{|desc_{L_1}^{level(c_1)}(c_1) \bigcap desc_{L_1}^{level(c_2)}(c_2)|}{|desc_{L_1}^{level(c_1)}(c_1) \bigcup desc_{L_1}^{level(c_2)}(c_2)|}$, *if* $c_1 = anc_{level(c_2)}^{level(c_1)}(c_2)$ *or* $c_1 = desc_{level(c_1)}^{level(c_2)}(c_2)$,

2. $dist_J(c_1, c_2) = \frac{|desc_{L_1}^{level(c_1)}(c_1)| - |desc_{L_1}^{level(c_2)}(c_2)|}{desc_{L_1}^{level(lca(c_1,c_2))}(lca(c_1,c_2))} + dist_S(c_1, c_2)$, *if* $level(c_1) = level(c_2)$,

3. ∞*, otherwise,*

where L_1 *is the most detailed level.*

Now, we define the Jaccard distance between states.

Definition 15 (Jaccard state distance) *Given two states* $s^1 = (c_1^1, c_2^1, \ldots, c_n^1)$ *and* $s^2 = (c_1^2, c_2^2, \ldots, c_n^2)$*, the Jaccard state distance* $dist_J(s^1, s^2)$ *is defined as*

$$dist_J(s^1, s^2) = \sum_{i=1}^{n} |dist_J(c_i^1, c_i^2)|.$$

4 Relaxed Context Resolution

The question that arises is which subset or subsets of the context parameters to relax and how much (i.e., what is an appropriate value for r) so that we have a large enough set of preferences in profile P that match the relaxed query. In this section, we highlight an implementation of relaxed context resolution that takes an exhaustive approach. We assume that the system (or the user) associates a value k with each query that specifies how many matching preferences should be returned. Given a contextual query Q with a context descriptor cod^Q, we search for k contextual preferences in P that match the set of context states specified by cod^Q. In cases, where there are not enough preferences, we gradually relax a number of the context parameter descriptors in cod^Q, by using larger relaxation depths.

4.1 Data Structures

To store the contextual preferences in P, we use a data structure called a *profile tree* proposed in [10]. Let P be a profile and P_C be the set of context states of all context descriptors that appeared 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 at most $n+1$ levels. Each one of the first n levels corresponds to a context parameter and the last one to the leaf nodes. 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 $key \in adom(C_k)$ 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 $adom(C_{k+1})$ 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 (state) leading to it. Fig. 2 depicts the profile tree for an instance of a profile P.

Let Q_C be the set of context states derived from the descriptor cod^Q of a contextual query Q. As opposed to [10], 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 represented by a data structure similar to the profile tree, that we call the

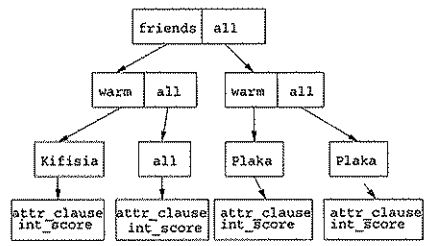

Figure 2: An instance of a profile tree.

Query tree, so that there is exactly one path in the tree for each context state $s \in Q_C$. Again, there is one level in the query tree for each context parameter.

4.2 A Context Resolution Algorithm

We describe first an algorithm that searches for states in the profile tree that are the same with the states of the query. In particular, given a *profile tree* whose root node is R_P and a *query tree* whose root node is R_Q, Algorithm 1 returns the leaf nodes of the *profile tree* that are associated with the paths whose context states match the querying context states. The profile tree has $n + 1$ levels. A context parameter is assigned to each one of the first n levels, while the last level holds the associated preferences. Respectively, the query tree has n levels, each for a context parameter. 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 ($i + 1$). When we check 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.

To support upwards, downwards and sideways relaxation, the *ContextResolution* algorithm is extended as follows. When the query result does not contain the desired number of answers, we relax the contextual query conditions in rounds. In particular, we search first for exact match states, then for relaxed states with distance equal to 1 from the searching states, then for relaxed states with distance equal to 2, and so on. In each round, i.e., for a specific distance value, we search first for upwards relaxed states, then, for downwards and finally for sideways relaxed states. The algorithm stops when the total number of returned context states is at least equal to the desired number of states.

Algorithm 1 ContextResolution Algorithm

1: **Input:** The *profile tree* with root node R_P and $n+1$ levels, and the *query tree* with root node R_Q and n levels.
2: **Output:** A *ResultSet* of tuples of the form (attr_name = attr_value, int_score) characterizing paths whose context states are the same with the searching context states, i.e., the states of the query tree.

SN, SN' sets of pairs of nodes.
Initially: $SN = \{(R_Q, R_p)\}$, $SN' = \{\}$

3: **Begin**
4: **for** level $i = 0$ to $n-1$ **do**
5: **for** each pair $sn \in SN$, with $sn = (q\_node, p\_node)$ **do**
6: $\forall x \in q\_node$
7: $\forall y \in p\_node$
8: **if** $x = y$ **then**
9: **if** $i < n-1$ **then**
10: $SN' = SN' \cup \{(x \rightarrow child, y \rightarrow child)\}$
11: **else if** $i = n-1$ **then**
12: $attr\_clause = y \rightarrow child.attr\_clause$
13: $int\_score = y \rightarrow child.int\_score$
14: $ResultSet = ResultSet \cup (attr\_clause, int\_score)$
15: **end if**
16: **end if**
17: **end for**
18: $SN = SN'$
19: $SN' = \{\}$
20: **end for**
21: **End**

4.3 Implementation Issues

Dewey Encoding To facilitate identifying descendants, ascendants and siblings values, assuming that each context parameter hierarchy is represented by a tree, we using Dewey encoding. A vector is assigned to each node of the hierarchy tree, starting from the root node. Each node vector is a combination of its parent vector and an integer number. If node n is the i^{th} child of node m with vector v_m, the vector of n is a concatenation of the vector of m and i, and represented as $v_n = v_m.i$. The Dewey encoding for the context parameter *weather* is depicted in Fig. 3.

Profile Pre-Processing Using Frequent Item Sets We apply a pre-processing technique to preferences in a profile, before indexing them using the profile tree. The goal is to determine which context parameter should be mapped to which level of the tree so that the size of the tree is kept small. Furthermore, assuming that pre-processing is based on the simple observation that if a parameter C_i is mapped to level i, it is useful to map a parameter C_j to level

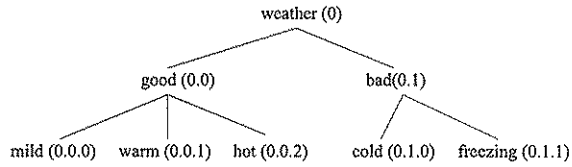

Figure 3: Dewey Encoding.

$i+1$, if many pairs of similar values (c_i, c_j), with $c_i \in C_i$ and $c_j \in C_j$ appear in the profile. Thus, to find an appropriate ordering of context parameters, we are interested in identifying the pairs of context values that are most frequent. To identify these pairs of values, we use the fundamental *a priori* property of frequent itemsets that states that every subset of a frequent itemset must also be a frequent itemset.

First, we identify the most frequent context values grouping them according to the context parameters that they belong to. Next, we search for the most frequent pairs of context values (c_i, c_j), with $c_i \in C_i$, $c_j \in C_j$ and $C_i \neq C_j$. In particular, considering that parameter C_i is placed to the level i of the profile tree, we search which parameter to place at level $i+1$. So, among the rest parameters, we select the parameter C_j, if the number of pairs of context values between C_i and C_j are more than the number of pairs between C_i and any other parameter. In general, we consider that a value or a pair of values is frequent, if it appears in contextual preferences more times than a threshold value. This threshold is different for singletons than for pairs of values, expressed as the percentage of either all single values or all pairs of values.

Statistics For each context value $c_i \in adom(C_i)$, we maintain the value $appearances(c_i)$, $appearances(c_i) = ap(c_i)/ap(all)$, where $ap(c_i)$ is the total number of times that the values of the leaf nodes with root c_i in the hierarchy of C_i appear in context states of profile P and $ap(all)$ is the total number of times that all leaf nodes appear in context states. We use these statistics to select between two context values the one that is more possible to appear in a context state of a preference.

5 Performance Evaluation

In this section, we present some 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 namely, 500, 1000, 3000, 5000, 8000 states. The values of two of the context parameters are drawn from their corresponding domain using a zipf data distribution with $a = 2.0$, while the values of the third parameter are selected using a uniform data distribution. Input parameters are summarized in Table 1. We stress out that our experimental results are meant to be indicative of the benefits

Table 1: Input Parameters

Context Parameters	Default Value	Range
Number of context parameters	3	
Profile size (context states)	5000	500-8000
Cardinality of domains	100	
Hierarchy levels	3	
Data distribution		
zipf	a = 2.0	
uniform		
Number of query states	20, 50	20, 50, 100

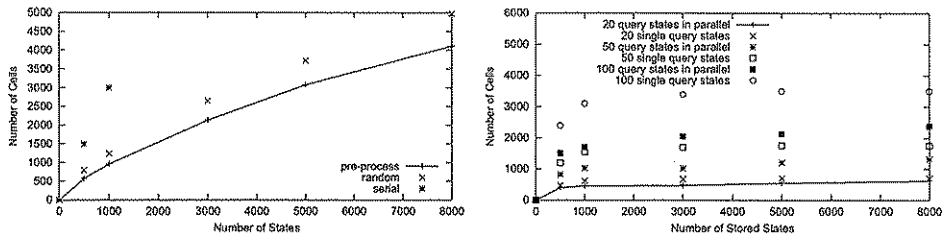

Figure 4: Size of the profile tree (left), number of cell accesses in exact match (right).

of some aspects of the proposed approach rather than forming an exhaustive performance evaluation.

Profile Pre–Processing In this set of experiments, we study the size of the profile tree as a function of the size of the profile (i.e., the number of its context states), when using the pre-processing technique to map context parameters to levels. As shown in Fig. 4 (left), using the pre-processing technique results in reducing storage by around 20% when compared to assigning parameters to levels randomly. We also show the cost of storing the states sequentially (without using the profile tree).

Context Resolution Using the Query Tree We perform a number of experiments to compare the performance of searching for matching states for each state of the query separately versus matching all query states in parallel using Algorithm 1. Fig. 4 (right) depicts our results for exact match queries having query descriptors with 20, 50 and 100 states. In each case, we count the total number of cells accessed. Searching for all states in one pass results in savings at around 35% on average.

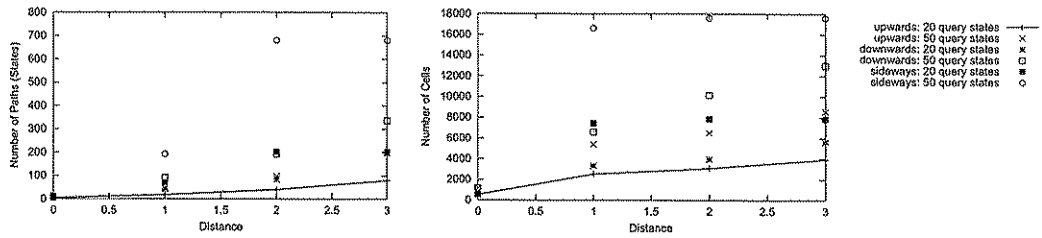

Figure 5: Number of returned states (left), number of cell accesses (right).

Relaxing Contextual Conditions In this set of experiments, we report the number of returned context states and the associated number of cell accesses, when we relax query context states. We run these experiments for profiles with 5000 context states. 75% of context values are considered to be values at the most detailed level of the hierarchies and the rest 25% are assigned to the other two levels. Fig. 5 (left) shows the number of returned context states for upwards, downwards and sideways relaxation, up to a distance 1, 2, 3, when we search for 20 and 50 states, using the query tree (parallel version of context resolution). We also report results (Fig. 5, right), regarding the number of cell accesses in each case. As expected, the upwards direction incurs less costs that the downwards one. Sideways relaxations incurs higher costs than both.

6 Related Work

The research literature on preferences is extensive. In the context of database queries, there are two different approaches for expressing preferences: a quantitative and a qualitative one. With the *quantitative approach* (i.e., [1, 7]), preferences are expressed indirectly by using scoring functions that associate a numeric score with every tuple of the query answer. In the *qualitative approach* (i.e., [2]), preferences between tuples in the answer to a query are specified directly, typically using binary preference relations. Relaxed context resolution applies equally to both approaches. There is some recent work on context-aware preferences. In our previous research [9, 10], we have addressed the problem of expressing contextual preferences. The model used in [9] 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 of single context parameters. In [10], we allow contextual preferences that involve more than one context parameter. Context as a set of dimensions (e.g., context parameters) is also considered in [8] where the problem of representing context-dependent semistructured data is studied. Contextual preferences, called situated preferences, are also discussed in [3]. Situations (i.e., context states) are uniquely linked through an N:M relationship with preferences expressed using the quantitative approach. A knowledge-based context-aware

query preference model is also proposed in [11].

Query relaxation has attracted some attention recently. A framework to relax queries involving numeric conditions in selection and join predicates is proposed in [6]. In this paper, we focus on categorical attributes with hierarchical domains. The relaxation algorithm proposed in [4] produces a relaxed query for a given initial range query and a desired cardinality of the result set. To estimate the query size, the algorithm uses multi-dimensional histograms. 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 attribute with each of its dimensions taking values from hierarchical domains. Each query is also augmented with a set of context states. We consider the problem of relaxing the context states of the query, so that there are enough preferences whose associated context states match that of the query. We consider relaxing the value of each dimension by either using a more general (upwards relaxation), a more specific (downwards relaxation) or a sibling (sideways relaxation) value. Depending on the distance in the associated hierarchy between the original and the relaxed value, we define different relaxation levels. This results in a large number of potential relaxations of context by appropriately relaxing different subsets of its dimensions with various relaxation depths and in any of the three ways. We present an algorithm that incrementally relaxes the query context, until a sufficiently large number of results is produced. We also present some initial performance results.

References

[1] Rakesh Agrawal and Edward L. Wimmers. A framework for expressing and combining preferences. In *ACM SIGMOD*, 2000.

[2] Jan Chomicki. Preference formulas in relational queries. *ACM Trans. Database Syst.*, 2003.

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

[4] A. Kadlag, A. V. Wanjari, J. Freire, and J. R. Haritsa. Supporting exploratory queries in databases. In *DASFAA*, 2004.

[5] Werner Kießling and Gerhard Köstler. Preference sql - design, implementation, experiences. In *VLDB*, 2002.

[6] Nick Koudas, Chen Li, Anthony K. H. Tung, and Rares Vernica. Relaxing join and selection queries. In *VLDB*, 2006.

[7] Georgia Koutrika and Yannis Ioannidis. Personalized queries under a generalized preference model. In *ICDE*, 2005.

[8] Y. Stavrakas and M. Gergatsoulis. Multidimensional Semistructured Data: Representing Context-Dependent Information on the Web. In *CAiSE*, 2002.

[9] Kostas Stefanidis, Evaggelia Pitoura, and Panos Vassiliadis. Modeling and storing context-aware preferences. In *ADBIS*, 2006.

[10] Kostas Stefanidis, Evaggelia Pitoura, and Panos Vassiliadis. Adding context to preferences. In *ICDE*, 2007.

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

[12] P. Vassiliadis and S. Skiadopoulos. Modelling and Optimisation Issues for Multidimensional Databases. In *CAiSE*, 2000.