Approximate Contextual Preference Scoring in Digital Libraries*

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

Abstract

The increasing availability of a wide variety of digital libraries provides a heterogeneous population of users with access to a heterogeneous collection of data. To help users find interesting items from this huge volume of available information, personalization systems allow users to directly or indirectly indicate preferences. Such preferences may depend on context, for instance, on the device used to access information. In this paper, we consider a preference model that allows users to assign interest scores to pieces of data based on context. We address the problem of locating interesting data items efficiently by exploiting some form of precomputation. To this end, instead of computing scores for all data items in all potential context states, we exploit the hierarchical nature of context attributes to identify representative context states and compute scores for them. This provides users with fast but potentially approximate results. We discuss this issue and report some preliminary results.

1 Introduction

Nowadays, a rapidly increasing number of organizations, such as libraries, museums and galleries, organize their collections electronically for making them available to the large number of Web users. Such collections have varying themes and may be represented in different ways. The work in [9] offers a detailed view on how to proceed from the current to the next generation of digital libraries. Integrating personalization and recommendation systems with digital libraries would greatly assist users in finding items of interest in this huge and diverse volume of information [16]. In such systems, users indicate preferences either directly or indirectly.

Various preference models have been proposed, most of which follow either a quantitative or a qualitative approach. With the quantitative approach (e.g., [2, 11]), users employ scoring functions that associate a numerical score with specific pieces of data to indicate their interest on them. With the qualitative approach (such as the work in [3, 10]), preferences are specified between two pieces of data, typically using binary preference relations. The work in [12] considers personalized query results in digital libraries focusing on posing queries without complete knowledge of the database schema. The answers of such queries contain not only items explicitly defined in the queries, but also items implicitly related to them. A recommender system based on user preferences is proposed in [15], where a framework is introduced for measuring similarity of qualitative preferences.

Motivated by the fact that preferences may depend on context, contextual preferences have been recently introduced for the quantitative [17, 18] and the qualitative [1, 8] approach. Knowledge-based contextual preferences have also been proposed [19]. Context is a general term used to express the situation of the user at the time of the submission of a query [4]. We use context in a broad sense to indicate any attribute that is not part of the database schema. To allow more flexibility in expressing preferences, we assume that context attributes take values from hierarchical domains.

We address the problem of ranking database tuples based on contextual preferences by using precomputed scores. Assuming that the database is large and only a few tuples are of interest at any given context state, precomputing a score for all database tuples for each potential context state would result in both wasting resources and slow query responses. Thus, instead, we precompute scores only for *interesting* tuples and for *representative* context states. Our method for computing representative context states exploits the hierarchical nature of context attributes for defining similarity between context states.

2 Contextual Preference Ranking

As a running example, we consider a single database relation with information about the contents of libraries, such as books and magazines with schema: *Content*(isbn, name, category, author, publisher, publication_date).

Contextual Preference Model. Context is modeled through a finite set of special-purpose attributes, called *context parameters* (C_i). Each context parameter C_i is characterized by a context domain $dom(C_i)$, that is, an infinitely countable set of values. In our running example, we consider three context parameters as relevant: $accompanying\_people$, $work\_environment$ and $time\_period$.

*Work partially supported by the Greek General Secretariat for Research and Technology through PENED 03-ED-591.

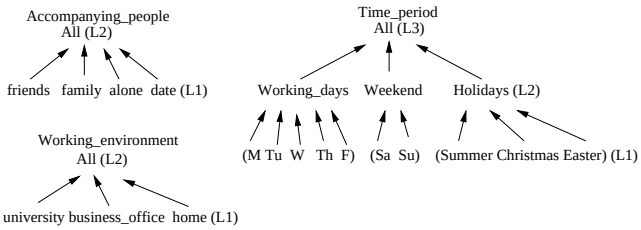

Figure 1. Context Hierarchies.

Similar to [18], we model context parameters using multi-dimensional hierarchical attributes. In particular, we assume that each parameter participates in an associated *hierarchy of levels* of aggregated data, i.e., it can be viewed from different levels of detail. We use the notation $dom_{L_j}(C_i)$ for the domain of level L_j of parameter C_i. A function $anc_{L_j}^{L_k}$ assigns a value of the domain of L_j to a value of the domain of L_k. For instance, $anc_{L_1}^{L_2}(summer) = holidays$. Furthermore, we use the notation $L_j \preceq L_k$ between two levels to mean $L_j \prec L_k$ or $L_j = L_k$. For n context parameters, a *context state* cs is a n–tuple of the form $(c_1, c_2, \ldots, c_n)$, where $c_i \in dom(C_i)$. Fig. 1 depicts the hierarchies of context parameters of our example. Such hierarchies may be constructed using for example the WordNet [14].

We use a simple quantitative preference model similar to the ones in [2, 18]. In particular, users express their preferences for sets of tuples specified using selection conditions on some of their attributes by providing a numerical score, which is a real number between 0 and 1. This score expresses a degree of interest. Value 1 indicates extreme interest, while value 0 indicates no interest.

Definition 1 (Contextual Preference) *Given a database schema $R(A_1, A_2, \ldots, A_d)$, a* contextual preference *p on R is a triple (cs, Pred, score), where cs is a context state, $Pred$ is a predicate of the form $A_{i_1}\theta_1 a_{i_1} \wedge A_{i_2}\ \theta_2\ a_{i_2} \ldots \wedge A_{i_k}\theta_k\ a_{i_k}$ that specifies conditions $\theta_i \in \{=, <, >, \leq, \geq, \neq\}$ on the values $a_{i_j} \in dom(A_{i_j})$ of attributes A_{i_j}, $1 \leq i_j \leq d$ and score is a real number between 0 and 1.*

The meaning of a contextual preference is that in context state cs, all database tuples t for which $Pred$ holds are assigned the indicated score. We use the notation $Pred[t]$ to denote that predicate $Pred$ holds for tuple t. We call the set of contextual preferences a *profile* P. By CS_P, we denote the set of context states cs that appear in at least one preference in P. Finally, we shall use the notation r to denote a database (instance) with schema R.

In a given profile, there may be no related preference for a tuple $t_i \in r$ in a context state cs. These tuples are assigned a default score of 0. This is because, we consider preferences expressed by users to be indicators of positive interest. Consequently, we assume that an unrated tuple is less important than any other tuple for which the user has expressed some interest.

In some cases, there may be more than one preference applicable to a specific database tuple, in the same context. Assuming two predicates $Pred_1$ and $Pred_2$, we say that $Pred_1$ subsumes $Pred_2$, iff $\forall\ t \in r$, $Pred_1[t] \Rightarrow Pred_2[t]$, i.e., we say that $Pred_1$ is *more specific* than $Pred_2$. To compute the score of each tuple at a given context state, we consider only the most specific predicates.

Definition 2 (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 $p_i = (cs,\ Pred_i,\ score_i)$, such that, $Pred_i[t]$ holds and $\neg\ \exists$ p_j, $p_j = (cs,\ Pred_j,\ score_j) \in P'$, such that, $Pred_j$ subsumes any $Pred_i$. The score of t in cs is:*

$$score(t, cs) = max_{p_i \in P'} score_i.$$

For example, assume the library relation of Table 1 and a profile with the following preferences: p_1 = *(family, category = graphic novel, 0.8)*, p_2 = *(family, author = Orwell, 0.7)*, p_3 = *(alone, category=fiction, 0.5)*, p_4 = *(alone, category = fiction $\wedge$ author = Miller, 0.8)*. In context $family$, for t_1 both preferences p_1 and p_2 are applicable. Similarly, both preferences p_3, p_4 are applicable to tuple t_2 in context $alone$. In the first case, no predicate subsumes the other and the score for t_1 is the maximum of the two scores, namely 0.8. Under context $alone$, the predicate of p_4 subsumes the predicate of p_3, and so, t_2 has score 0.8.

Assume a profile P and a database instance r. In general, when a query q is submitted, we would like to provide the user with those tuples t in r that have the largest scores for the context of q. Implicitly, the context of a query is the current context, that is, the context surrounding the user at the time of the submission of the query. Context may be also specified explicitly as part of the query. Many times, there are no preferences whose context state matches that of the query context. In this case, we would like to use preferences that refer to similar context states. We extend first the definition of a tuple score to include scores applicable to more than one context state.

Definition 3 (Aggregate Tuple Score) *Let P be a profile, $CS \subseteq CS_P$ be a set of context states, and $t \in r$ a tuple, $score(t, CS) = max_{cs \in CS} score(t, cs)$.*

Now, given that the distance between two context states, $dist_S$, is defined (see Section 3 for our definition), the problem is the following:

Problem Definition. Assume a database instance r, a profile P and a query q with context state cs_q. Let $CS_q \subseteq CS_P$ be the set of context states cs with the minimum $dist_S(cs_q, cs)$, that is the context states that are the most similar to the context state of the query. The contextual scoring problem is to find the top-k tuples $t \in r$ with the highest scores, computed as: $ag\_score(t, cs_q) = score(t, CS_q)$.

A solution that involves no pre-computation is to first find the set of context states CS_q, compute the aggregate scores of all tuples $t \in r$ and return the top-k of them. Performance can be improved by performing preprocessing steps offline. One approach would be to compute the scores of each tuple

Table 1. Database Instance

	$ISBN$	$Name$	$Category$	$Author$	$Publisher$	$PublicationDate$
t_1	0451526341	Animal Farm	Graphic Novel	George Orwell	Signet Classics	1996
t_2	1569714029	300	Fiction	Frank Miller	Dark Horse	2000

for each potential context state. Assuming a large database and that only a few tuples are of interest at any given context, finding the top-k tuples for all database tuples for each context state will result in both wasting resources and slow query responses. Since, the number of possible context states grows exponentially with the number of context variables, we could instead compute the scores for all states that appear in the profile and then combine the scores of the most similar ones online.

Since the number of context states that appear in a profile can still be large, in this paper we consider an approach for finding representative scores to precompute. Our approach constructs clusters of preferences, considering as similar the preferences that have either the same or similar context states. After constructing the clusters of preferences, we compute for each of them an interest score for each database tuple that is applicable to at least one preference of the cluster, using the relative to the cluster preferences. Furthermore, instead of storing scores for all tuples for each cluster, we just store the scores that are larger than 0. Then, for a submitted query, we search for the most similar to the query cluster and provide quickly the top-k results, that is, the k database tuples with the highest scores.

3 Similarity between Context States

To define the distance between two context states, we define first the distance between two context values. One direct method to compute such a distance is to find the length of the minimum path that connects them in their associated hierarchy. However, this method may not be accurate, when applied to attributes with large domains and many hierarchy levels (e.g., smaller path lengths for less similar values). For instance, in our simple example of $Time\_period$ hierarchy, taking into account only the path length, we observe that the pair of values $Christmas$, $Easter$ has the same distance with the pair $Christmas$, All, while it would probably make sense for $Easter$ to be more similar to $Christmas$ than All. Motivated by related research on defining semantic similarity between terms (e.g., [13]), to compute the distance, we also take into account the depth of the hierarchy levels that the two values belong to. We define first the *path distance*. Let $lca(c_1, c_2)$ be the least common ancestor of context values c_1 and c_2.

Definition 4 (Path distance) *The path distance* $dist_P(c_1, c_2)$ *between two context values* $c_1 \in dom_{L_j}(C_i)$ *and* $c_2 \in dom_{L_k}(C_i)$*:*

- *is equal to 0, if* $c_1 = c_2$,
- *is equal to 1, if* c_1, c_2 *are values of the lowest hierarchy level and* $lca(c_1, c_2)$ *is the root value of their corresponding hierarchy,*
- *or is computed through the* f_p *function* $(1 - e^{-\alpha \times \rho})$*, where* $\alpha > 0$ *is a constant and* ρ *is the minimum path length connecting them in the associated hierarchy.*

The f_p function is a monotonically increasing function that increases as the path length becomes larger. Furthermore, the above definition of *path distance* ensures that the distance is normalized in $[0, 1]$.

Definition 5 (Depth distance) *The depth distance* $dist_D(c_1, c_2)$ *between two context values* $c_1 \in dom_{L_j}(C_i)$ *and* $c_2 \in dom_{L_k}(C_i)$*:*

- *is equal to 0, if* $c_1 = c_2$,
- *is equal to 1, if* $lca(c_1, c_2)$ *is the root value of their corresponding hierarchy,*
- *or is computed through the* f_d *function* $(1 - e^{-\beta/\gamma})$*, where* $\beta > 0$ *is a constant and* γ *is the minimum path length between the* $lca(c_1, c_2)$ *value and the root value of the corresponding hierarchy.*

The f_d function is a monotonically increasing function of the depth of the least common ancestor. Again, the definition of *depth distance* ensures distances within the range $[0, 1]$. Having defined the path and the depth distances between two context values, we define next their overall distance.

Definition 6 (Value distance) *The value distance between two context values* c_1 *and* c_2 *is computed as* $dist_V(c_1, c_2) = dist_P(c_1, c_2) \times dist_D(c_1, c_2)$.

For example, assuming the values $working\_days$ and $summer$, their path distance is $1 - e^{-3} \simeq 0.95$, their depth distance is 1, and so, their distance value is $1 \times 0.95 = 0.95$. Given now, the values $holidays$ and $summer$ their distance is $(1 - e^{-1 \times 1}) \times (1 - e^{-1/1}) \simeq 0.39$, that means that the value $summer$ is more closely related to $holidays$ than to $working\_days$. In both examples, we assume that $\alpha = \beta = 1$.

Note that to compute the value distance $dist_V$, we use the independent $dist_P$ and $dist_D$ distances. This independency enables us to combine them in different ways by giving different weights of interest. To do this, we may assign different values to the constants α, β. In particular, for constant values greater than 1, distance increases, while values within the range (0, 1) result in smaller distances.

Having defined the distance between two context values, we can now define the distance between two context states.

Definition 7 (State distance) *Given two context states* $cs^1 = (c_1^1, c_2^1, \ldots, c_n^1)$ *and* $cs^2 = (c_1^2, c_2^2, \ldots, c_n^2)$*, the state distance is defined as:*

$$dist_S(cs^1, cs^2) = \sum_{i=1}^{n} w_i \times dist_V(c_i^1, c_i^2),$$

where each w_i *is a context parameter specific weight.*

The above weights are normalized, such that, $\sum_{i=1}^{n} w_i = 1$. Each weight takes a value according to the cardinality of its related context parameter domain. In particular, we assign larger weights to parameters with smaller domains, considering a higher degree of similarity among values that belong to a large domain.

It is easy to show that the distance relationship between context states is reflexive ($dist_S(cs_1, cs_1)$ = 0), and symmetric ($dist_S(cs_1, cs_2)$ = $dist_S(cs_2, cs_1)$). However, it does not satisfy the triangle inequality ($dist_S(cs_1, cs_2) \leq dist_S(cs_1, cs_3)$ + $dist_S(cs_3, cs_2)$), because of the semantic way of defining distances among context values.

4 Contextual Clustering

To group preferences with similar context states, we use the *d-max* algorithm (shown in Algorithm 1). The *d-max* algorithm is a hierarchical clustering method that follows a bottom-up strategy. Initially, each context state is placed in its own cluster. At each step, the algorithm merges the two clusters with the smallest distance. The distance between two clusters is defined as the maximum distance between any two context states that belong to these clusters. The algorithm terminates when the closest two clusters, i.e., the clusters with the minimum distance, have distance greater than d_{cl}, where d_{cl} is an input parameter. Finally, for each produced cluster, we select as representative context state, the state in the cluster that has the smallest total distance from all the states of its cluster. Formally:

Definition 8 (Representative context state) *Let* cl_i *be a cluster produced by the d-max algorithm that consists of a set* W_{cl_i} *of* m *context states,* cs_{i_j}*. The representative of* cl_i *is the context state* $cs \in W_{cl_i}$*, with the minimum distance* $\sum_{j=1}^{m} dist_S(cs, cs_{i_j})$.

Using the *d-max* algorithm, any two context states cs_1, cs_2 that belong to the same cluster have distance $dist_S(cs_1, cs_2) \leq d_{cl}$.

Having created the clusters of preferences, we compute for each of them an aggregate score for each tuple specified in any of its preferences (using the definition of the aggregate tuple score). For each produced cluster cl_i, we maintain a relation table $cl_i Scores$(tuple_id, score), in which we store in decreasing order only the scores of tuples that satisfy at least one of the predicates in the preferences of the cluster. That is, we do not maintain scores for all tuples, but only for those having nonzero scores, keeping in mind, that the remaining ones have score equal to 0. Each time a query is submitted, we search for the most similar cluster or clusters, that means, for the clusters whose representative context state is the most similar to the query context state. Then, using the relative to the clusters table of scores, we retrieve the k tuples with the highest scores.

Algorithm 1 *d-max* Algorithm

Input: A set of preferences with context states cs_i, a distance value d_{cl}.
Output: A set of clusters.

Begin
1. Create a cluster for each context state cs_i.
2. Repeat.
   2.1 If the minimum distance among any pair of clusters is smaller than d_{cl}.
      2.1.1 Merge these two clusters.
   2.2 Else, end loop.
3. Compute the representative context state of each produced cluster.

End

It is straightforward (by Definition 3) that:

Property 1 *Let a context state* cs *and a set of context states* CS*. 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 the representative context state is no less than the score of the tuple computed using any of the context states belonging to the cluster. In other words, if the context state that is the most similar to the query context belongs to the cluster whose representative context state is the most similar to the query, then the score that our approximation approach computes for a tuple cannot be lower than the exact one. That is, we may overate a tuple, but never underrate it.

To guarantee that the most similar context states belong to the selected cluster, we may need to check more than one cluster (not just the ones whose representative context state is the most similar to the query state). The following property indicates how (proof omitted due to space limitations).

Property 2 *Assume a query* q *with context state* cs_q*. Let* cs_p *be the most similar to* cs_q *context state and let* $dist_S(cs_q, cs_p)$ = d*. To guarantee that* cs_p *belongs to the clusters selected for computing the answers of* q*, we need to return all clusters with representative context state, say* cs_r*, such that,* $dist_S(cs_q, cs_r) \leq d + d_{cl}$*, where* d_{cl} *is the input distance parameter of the d-max algorithm.*

When more than one cluster are used to compute the $top-k$ results of a query, we can apply a $top-k$ algorithm (such as, FA, TA or their variations [6, 5, 7]) to the relation tables $cl_i Scores$ of the related to the query clusters.

5 Performance Evaluation

Clearly there is a trade-off between the number of clusters and the quality of the produced scores. To evaluate our approach, we run a set of preliminary experiments using a database with 10000 tuples and 1000 contextual preferences.

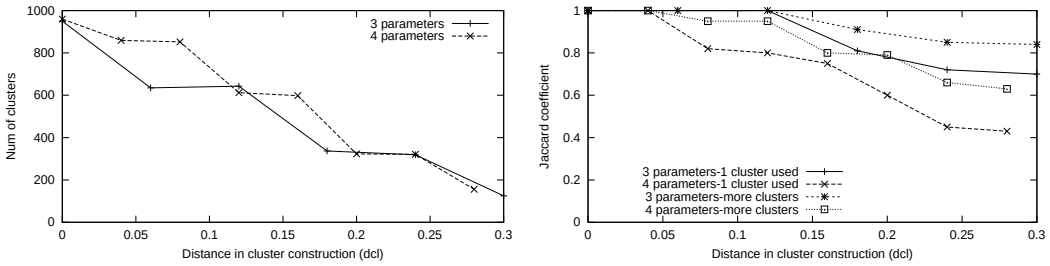

Figure 2. (Left) number of clusters and (right) result quality for different d_{cl} values

Context values are selected using a zipf data distribution with a = 1.5, from context domains with 100 values and 4 hierarchy levels.

In Fig. 2 (left), we present the number of produced clusters with regards to the parameter d_{cl} of the *d-max* algorithm, when preferences are expressed with 3 or 4 context parameters. As expected, a large d_{cl} value results in a small number of clusters. Since we keep tuple scores only for the produced clusters, this is an indication of the amount of precomputed information that we maintain.

We also test the quality of the top-k results. In particular, assume that $Results(d\text{-}max)$ is the set of the top-k tuples computed using the *d-max* algorithm and $Results(opt)$ is the set of top-k tuples computed using the context states that are most similar to the query. We compare these two sets using the Jaccard coefficient defined as: $\frac{|Results(d-max) \cap Results(opt)|}{|Results(d-max) \cup Results(opt)|}$. We consider the following two cases: (i) we use only the most similar clusters, and (ii) we use all the clusters that are necessary to guarantee that the most similar to the query states belong to one of them, according to Property 2. The Jaccard coefficient takes values between 0 and 1: the higher its value, the more similar the two top-k tuple sets. We report the results for k = 20 (Fig. 2, (right)). Finally, note that if the set of preferences used to compute the top-k results is selected randomly, the Jaccard coefficient is nearly 0.

6 Summary

Digital libraries offer to a heterogeneous population of users efficient access to a heterogeneous population of data items distributed across the Web. In this paper, we consider enhancing such information systems with context dependent preferences. Context is modeled as a multidimensional attribute with each of its dimensions taking values from hierarchical domains. We address the problem of finding interesting data items based on preferences that assign interest scores to pieces of data based on context. To this end, instead of computing scores for all data items in all potential context states, we exploit the hierarchical nature of context attributes to identify representative context states.

References

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

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

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

[4] A. K. Dey. Understanding and Using Context. *Personal and Ubiquitous Computing*, 2001.

[5] R. Fagin. Combining fuzzy information from multiple systems. *J. Comput. Syst. Sci.*, 58(1):83–99, 1999.

[6] R. Fagin, A. Lotem, and M. Naor. Optimal aggregation algorithms for middleware. In *PODS*, 2001.

[7] U. Güntzer, W.-T. Balke, and W. Kießling. Optimizing multifeature queries for image databases. In *VLDB*, 2000.

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

[9] Y. E. Ioannidis. Digital libraries at a crossroads. *Int. J. on Digital Libraries*, 5(4):255–265, 2005.

[10] W. Kießling and G. Köstler. Preference sql - design, implementation, experiences. In *VLDB*, 2002.

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

[12] G. Koutrika and A. Simitsis. An enhanced search interface for information discovery from digital libraries. In *ECDL*, 2006.

[13] Y. Li, Z. A. Bandar, and D. McLean. An approach for measuring semantic similarity between words using multiple information sources. *IEEE TKDE*, 15(4):871–882, 2003.

[14] G. A. Miller. Wordnet: a lexical database for english. *Commun. ACM*, 38(11):39–41, 1995.

[15] B. Satzger, M. Endres, and W. Kießling. A preference-based recommender system. In *EC-Web*, pages 31–40, 2006.

[16] A. F. Smeaton and J. Callan. Personalisation and recommender systems in digital libraries. *Int. J. on Digital Libraries*, 5(4):299–308, 2005.

[17] K. Stefanidis, E. Pitoura, and P. Vassiliadis. Modeling and storing context-aware preferences. In *ADBIS*, 2006.

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

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