Preference-Based Queries for Course Sequencing

Periklis Georgiadis
Dept. of Computer Science
University of Crete, Heraklion, Greece
perge@csd.uoc.gr

Vassilis Christophides
Institute of Computer Science
FORTH, Heraklion, Greece
christop@ics.forth.gr

Nicolas Spyratos
Laboratoire de Recherche en Informatique, Université Paris Sud
Orsay Cedex, France
spyratos@lri.fr

Abstract[1]**:** We address the problem of generating entire course sequences, given a set of target skills together with possibly prioritized preferences over course descriptions. The objective is to support students in defining personalized curricula.

1. Introduction

Higher education curricula are characterized nowadays by their continuous evolution as well as by the mobility of students across them. Consequently, students are striving for support in choosing an appropriate course sequence that leads to a set of desired skills. Here, by "skill" we mean the ability (usually learned) to perform actions. In this paper we are interested in providing a student with a course sequence that (a) leads to the acquisition of the desired skills and (b) respects preferences expressed by the student as well as possible institutional restrictions (e.g. prerequisites).

In the context of our work, a skill describes the distinct effects of a course on the cognitive and technical level of the student. A skill can be either atomic or compound. An *atomic skill* is acquired by following a single course, although a single course might lead to one or more atomic skills. A *compound skill* is a combination of other, atomic or compound skills. We assume that skills are organized into a (part-of) taxonomy, the *skill taxonomy*, where the leaves are atomic skills, while all others are compound.

We consider courses as individual objects and assume that each leads to one or more atomic skills. The set of skills to which a course leads is just one of the attributes in a course description. Other attributes of a course are the period of the year in which the course is offered, the set of courses for which it is a prerequisite, and so on. The descriptions of all courses offered by an educational institution constitute what we call the *course catalogue*.

In our context, a student request for a personalized course sequence has the form of a query over the course catalogue and consists of two parts: (a) a filtering condition - usually represented by the set of desired skills, and (b) a number of binary relations, expressing student preferences and/or curriculum restrictions. We call such queries *preference-based queries*.

Motivating Example. To illustrate our general approach, consider Zoe, a student who wishes to acquire a given set of skills, and suppose that the desired skills require her to follow a set of courses C ={c1,c2,c3,c4,c5,c6,c7}. This set is actually determined through the course catalogue in a way that we shall see later. Suppose that courses c1, c2, c3 and c4 are offered only during the spring term, whereas c5, c6 and c7 only during the fall term. As a result, the set C is partitioned into two blocks, the block of spring courses B_S={c1,c2,c3,c4} and the block of fall courses B_F={c5,c6, c7}. Let's call this partition π_T, so we have: π_T={B_S,B_F}. Clearly, π_T expresses the curriculum restriction over the set C of courses that the student is required to follow.

Now, suppose that Zoe prefers to start with spring courses. Her preference can be incorporated into π_T by ordering its blocks, that is by considering the block B_S as *preceding* block B_F. This results to the following *ordered partition* expressing both the curriculum restriction *and* the student preference (the arrow → is read as "precedes"):

$$[\pi_T, \rightarrow] = \{B_S \rightarrow B_F\} \quad (1)$$

Suppose next that for various reasons Zoe prefers to follow certain courses before others. These reasons might be her previous experience or background knowledge regarding some of the courses, or simply a matter of taste. For the sake of our example, suppose that Zoe prefers to follow c1, c2 and c7 before c3; c4 before c1 and c2; c5 before c1 and c7, and c6 before c2 and c7. These preferences can be conveniently described by a directed (acyclic) graph, as shown in Fig.1a. It is not difficult to observe that they induce a new ordered partition of the set C of courses, this time into three blocks, the first of which is B_1={c4,c5,c6}, the second B_2={c1,c2,c7}, and the third B_3={c3}. Let's call this partition π_P, so we have:

$$[\pi_P, \rightarrow] = [B_1 \rightarrow B_2 \rightarrow B_3] \quad (2)$$

In each block of this partition, no course is preferred to no other course in the block (so the courses of each block can be taken in parallel). Moreover, for each course of the second block there is a preferred one in the first block, and for each course of the third block there is a preferred one in the second block. Stated more generally, for each course of a block (other than the first block) there is a preferred course in the previous block.

Finally, suppose Zoe wishes to obtain a personalized course sequence capturing both of her personal preferences (i.e. starting in the spring term *and* the partial ordering of Fig.1a). The question then is how to combine the ordered partitions $[\pi_T,\rightarrow]$ and $[\pi_P,\rightarrow]$ to produce an ordered partition $[\pi,\rightarrow]$, whose ordering respects the orderings of *both* $[\pi_T,\rightarrow]$ and $[\pi_P,\rightarrow]$. The partition $[\pi,\rightarrow]$ will then be the required sequencing of C. Furthermore, a student might wish to attach weights, or *priorities* to her preferences. For example, Zoe might consider starting in the spring term as more important to her than the ordering expressed in Fig.1a. Clearly, different priorities over Zoe's preferences lead to different course sequences. For example, if she considers starting in spring more important than the ordering of Fig.1a, then our approach will produce the following ordered partition π:

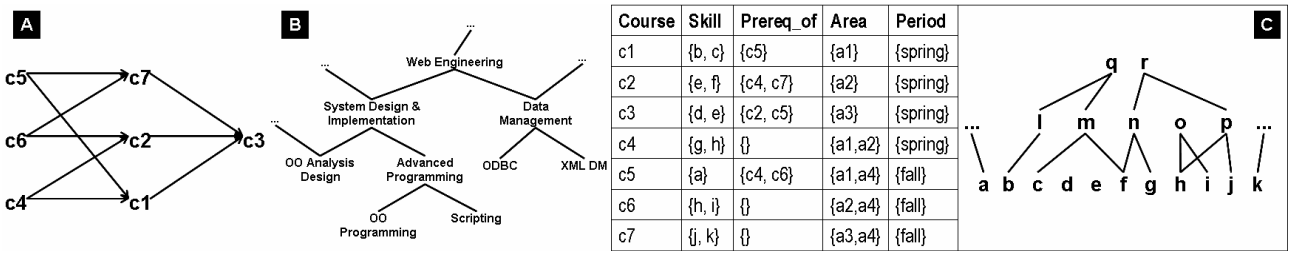
$$[\pi, \rightarrow] = [\{c4\} \rightarrow \{c1, c2\} \rightarrow \{c3\} \rightarrow \{c5, c6\} \rightarrow \{c7\}] \quad (3)$$

If however Zoe considers both preferences equally important, a different ordered partition is produced:

$$[\pi, \rightarrow] = [\{c4\} \rightarrow \{c1, c2, c5, c6\} \rightarrow \{c3, c7\}] \quad (4)$$

The main contribution of our work is the introduction of various ways of combining ordered partitions, while respecting (a) curriculum restrictions (b) student's

[1] This work is partially supported by the EU DELOS Network of Excellence in Digital Libraries (NoE-6038-507618).



Course	Skill	Prereq_of	Area	Period
c1	{b, c}	{c5}	{a1}	{spring}
c2	{e, f}	{c4, c7}	{a2}	{spring}
c3	{d, e}	{c2, c5}	{a3}	{spring}
c4	{g, h}	{}	{a1,a2}	{spring}
c5	{a}	{c4, c6}	{a1,a4}	{fall}
c6	{h, i}	{}	{a2,a4}	{fall}
c7	{j, k}	{}	{a3,a4}	{fall}

Figure 1: (a) A partial preorder on courses generated by Zoe's preferences, (b) An excerpt from a Skill Taxonomy in the IT Field, and (c) An excerpt of a Course Catalogue and a Skill Taxonomy (used in our examples)

preferences and (c) student's priorities over preferences. Although we do not consider monitoring of students' actual state of knowledge, or cognitive characteristics [3, 8,9,13,15,17,23,29], we are able to capture various aspects of course descriptions under the form of taxonomies and set-valued attributes, as well as student preferences under the form of binary relations. We are interested in generating entire course sequences prior to actual course attendance. Compared to logic frameworks addressing course sequencing as a planning problem [1,4,16,19], our work advocates a set-theoretic framework for generating course sequences using preference-based queries. To the best of our knowledge this is the first work in this area to employ such an algebraic approach. Compared to qualitative preference approaches [5-7,10-11,18,21,22,28], our work addresses fusion of ordered partitions of objects which originate from preferences expressed over general relations, rather than on functional attributes describing database tuples or objects. We believe that the proposed framework is expressive enough to produce course sequences from descriptions expressed in diverse data models (e.g., XML, RDF/S) with respect to a variety of user preferences, also including preference priorities.

The rest of the paper is organized as follows. In Section 2 we introduce the main concepts and assumptions of our approach. We present preference-based queries in Section 3, and we discuss how to combine preferences to produce personalized course sequences in Section 4. We finally conclude along with future work perspectives in Section 5.

2. Skill Taxonomies and Course Catalogues

As mentioned in the introduction, the focus of this paper is on skills, for which we assume a taxonomy capturing different levels of skill granularity (e.g. see Fig.1b).

Definition 1 (Skill Taxonomy): *A skill taxonomy is a reflexive, transitive binary relation over a set of skills S.*

Assuming that no two names are synonyms for the same skill, a skill taxonomy is a partial order, and the atomic skills are its minimal elements. It is best depicted by a Hasse diagram [14]: atomic skills are at the bottom and the higher the level the more compound the skills.

A curriculum offered by an educational institution is essentially a catalogue containing all courses offered along with their descriptions. Each course is described by a set of attributes, such as the period of the year (term) it is offered in, the set of courses it is a prerequisite for, the set of atomic skills it leads to, etc.

Definition 2 (Course Catalogue): *A course catalogue is a collection of course descriptions <c,d1,d2,...,dn> where c is a unique course identifier and each di is a set of values called the i-th description of c.*

The table in Fig.1c shows a course catalogue. Each row corresponds to the description of a single course. For example, the third row describes course c3, which leads to (atomic) skills d and e, is a prerequisite for c2 and c5, belongs to the thematic area a3, and is given in spring[2].

We assume that the entries of each column in the course catalogue are sets of atomic values that we call *terms*. Note that the empty set of terms {} might be used as an entry in the course catalogue. For example, in the table of Fig.1c, course c4 is not a prerequisite for any other course and thus the column `Prereq_of` is {}.

In what follows, we assume that each column is associated with a (predefined) set of terms from which the entries of the column are built. For example, the terms of column `Period` are fall, spring, …, the terms of column `Prereq_of` are c1, c2, …, and so on. The notation *i:t* denotes that *t* is a term of column *i*. So, `Period:fall` means that `fall` is a term of column `Period`, whereas `Skill:g` means that `g` is a term of column `Skill`, etc.

Terms can be viewed as keywords for searching the course catalogue. As such, they form the basis for defining a simple query language in which a query q can be a single term or a Boolean combination of terms:

`q::= i:t | q1 ∧ q2 | q1 ∨ q2 | q1 ∧ ¬q2` **(5)**

The answer of a query q, denoted by ans(q) is defined as follows: if q is a single term, say q=`i:t`, then its answer is the set of those courses whose description `di` contains t, i.e., ans(q)={*cj: t ∈ cj.i}*; otherwise, the answer is obtained by replacing each term `i:t` appearing in q by its answer set *{cj: t∈cj.i}*, and performing the set theoretic operations corresponding to the Boolean connectives. For example, in Fig.1c, query `Period:fall` will return the set of courses {c5,c6,c7}, whereas `Skill:h` set {c4,c6} and query `(Period:fall)∧(Skill:h)` set {c6} (i.e., the intersection of the previous two sets).

We assume that a course catalogue satisfies the following natural constraints:

1. all skills appearing in the catalogue are atomic skills (i.e. a compound skill can be acquired only by acquiring its constituent atomic skills)
2. no course can be in the catalogue unless at least one skill is specified as its outcome (i.e. the empty set can *not* be a value for the `Skills` attribute)
3. no relationship exists among skills of the same course (i.e. skills are acquired only on course completion)

Our running example comprises the course catalogue and the skill taxonomy depicted in Fig.1c.

3. Preference-based Queries over Course Catalogues

A *preference* over a column C of the course catalogue is a declaration of the form t → t′, where t and t′ are terms of C and where the arrow means that t is preferred to t′. For example, the declaration `spring→fall` is a preference

[2] It should be clear that the tabular representation of the course catalogue here is just one of the possible representations which also include ODMG, XML, or RDF/S.

over the column `Period` meaning that `spring` is preferred to `fall`. The set of all preferences declared over a column C is called a preference relation over C, and it is required to satisfy certain conditions:

Definition 3 (Preference relation): *Let C be a column of the course catalogue. A preference relation over C is any reflexive and transitive binary relation over the terms of C.*

A preference relation expresses a student's preferences over the terms of some column of the course catalogue. However, some of the columns of the course catalogue may describe regulations or restrictions that the educational institution imposes on the curricula and which can *not* be overridden by student preferences (e.g. no preference can be expressed over the column `Prereq_of`). In other words, the student is authorized to declare preferences over some columns only. We shall refer to these columns as *preference columns*.

Let → be a preference relation over the course catalogue. We call two terms t, t′ *equivalent* w.r.t. → if both t→t′ and t′→t hold. With this definition at hand the preference relation → becomes a partial order over equivalence classes of terms. Hereafter, we shall talk of terms up to equivalence.

Definition 4 (Preference query): *A preference query is a pair Q= <S, →>, where S is a set of (taxonomy) skills and → is a preference relation over the course catalogue.*

To simplify notation we shall denote a preference relation by the column name followed by the set of preferences over terms of that column. For example, the notation `AREA`: a1→a3, a2→a3 denotes a preference relation over `AREA` containing two preferences, a1→a3 and a2→a3.

In our approach, a curriculum personalization system comprises a skill taxonomy and a course catalogue (such as those of Fig.1c). Students interact with the system by submitting preference queries. When a preference query Q= <S, →> is submitted to the system, the system evaluates the query and returns to the student a *personalized curriculum* that is a set of courses partitioned into a set of linearly ordered blocks such that:

- all courses of a block can be taken in parallel.
- each block of courses should be taken in the specified block order;
- the block order respects all query preferences.

To define a preference query the student is presented with two menus, the *skill menu* and the *preference menu*. The former contains all skill names appearing in the skill taxonomy. The student defines the set S of the query by selecting the desired skill names from the menu. The latter menu contains the names of all preference columns (i.e. those columns over which preferences can be expressed). To define a preference relation the student first selects a column in the preference menu; after selection, a sub-menu appears with the terms of the selected column; the student then specifies a preference relation by giving a set of pairs of the form (t, t′), where t, t′ are terms from the sub-menu.

When a preference query Q= <S, →> is defined and submitted to the system its evaluation follows two steps:

1. determining the set of courses to be taken by the student
2. sequencing these courses to obtain the personalized curriculum.

3.1 Determining the set of courses

To determine the set of courses in the answer of Q= <S, →>, the system performs two operations, *initial course selection* and *course completion*.

Initial course selection

1. For each skill s in S the system determines the set of its constituent atoms, let it be denoted atoms(s), by traversing the skill taxonomy. Define:

$$\text{atoms}(S) = \cup\{\text{atoms}(s): s \in S\} \quad (6)$$

Let atoms(S)={t1, t2, …,tm}.

2. Define the following query over the course catalogue:

$$q = (\texttt{Skill:t1}) \vee (\texttt{Skill:t2}) \vee \ldots \vee (\texttt{Skill:tm}) \quad (7)$$

Let C be the set of courses returned by q.

Course completion

The set C returned by the previous step is now completed by taking into account course prerequisites. Course completion can be performed by a transitive closure computation over the prerequisite relation [12].

Example: In the context of our running example, consider the following preference query Q:

`S= {f, p, n}` with `AREA: a1→a3, a2→a3` (**8**)

During initial course selection, the system first determines the atoms of S, based on the skill taxonomy, and then forms a query over the course catalogue by taking the disjunction of all atoms of S:

$$\text{atoms}(S) = \{f, g, h, j\} \quad (9)$$

$$q = (\texttt{Skill:f}) \vee (\texttt{Skill:g}) \vee (\texttt{Skill:h}) \vee (\texttt{Skill:j}) \quad (10)$$

The evaluation of q over the course catalogue returns the following set of courses:

$$C = \{c2, c4, c6, c7\} \quad (11)$$

During course completion, the algorithm takes C as input and returns the following (completed) set C:

$$C = \{c1, c2, c3, c4, c5, c6, c7\} \quad (12)$$

3.2 Sequencing the set of courses

Initial course selection and course completion returns the set of courses that should be taken by the student to acquire the desired skills while satisfying the prerequisite relation. However, this set of courses is unordered. The next step in the evaluation of a preference query is to order this set of courses based on the preference relation of the query. This is done in three steps as follows:

1. *Ordering the course descriptions*: we use the preference relation → over terms to define a preference relation over the descriptions of the courses in C
2. *Ordering the courses*: we use the preference relation over descriptions to define a partial order over the courses in C
3. *Answering the preference query*: we use the partial order over C to construct a linearly ordered partition of C, and this partition is defined as the answer of Q

Ordering the descriptions: Based on the preference relation → of Q we can define a preference relation over descriptions in one of several ways (see for example [17, 26,27]). For the purposes of this paper, we define it as stated in the following proposition.

Proposition 1 (Ordering the descriptions): *Let → be a preference relation over a column of the course catalogue. Define the relation → over descriptions (of that same column) as follows:*

$d \rightarrow d'$ iff $\forall t' \in d'\ \exists t \in d$ such that $t \rightarrow t'$, for all descriptions d, d′ (of that column)

Then → is a preference relation over descriptions[3].

[3] This is a well known powerdomain order relation, other examples of which are the database sub-typing relation, and the Smyth powerdomain relation.

As in the case of terms, we can define two descriptions to be equivalent if both $d \rightarrow d'$ and $d' \rightarrow d$ hold. With this definition at hand the preference relation $\rightarrow$ becomes a partial order over equivalence classes of descriptions.

Ordering the courses: Based on $\rightarrow$ we now define a partial order over courses as stated below:

Proposition 2 (Ordering the courses): *Let $\rightarrow$ be a partial order over the descriptions in a column of the course catalogue. Define the relation $\rightarrowtail$ over courses as follows:*

$$c \rightarrowtail c' \text{ iff } c.d \rightarrow c'.d', \text{ for all courses } c, c'$$

Then $\rightarrowtail$ is a partial order over courses (up to equivalence of descriptions).

Answering the preference query: We now use the partial order $\rightarrowtail$ over C (as defined in Prop. 2) to define a linearly ordered partition of C; the latter will be the answer to Q.

(C, $\rightarrowtail$) is mapped to a graph G, whose nodes are the courses in C, and its arrows represent the $\rightarrowtail$ relation. As $\rightarrowtail$ is a partial order (up to equivalence), graph G is acyclic, therefore it has one or more roots (minimal elements). For any non-root node c of G, we define the length of the longest path from any root to c as the *distance* of c (and we set 0 as the roots' *distance*). Let m be the longest such *distance*. We have an algorithm called ORDPART (a straight-forward variant of topological sorting) to define a linearly ordered partition $[\pi, \rightarrow] = [B0 \rightarrow B1 \rightarrow \ldots \rightarrow Bm]$ of G's nodes; ORDPART simply iterates m+1 times on G; in each iteration i it extracts G's current roots into a set of nodes (block) Bi. It is not difficult to see that $[\pi, \rightarrow]$ is indeed a linear order (i.e. block Bi precedes Bi+1), and that the following facts hold:

- every course (node) of G is in one and only one block;
- in each block Bi, other than B0, each course has at least one, more preferred course in the block Bi-1, i=1,…, m;
- in each block, no course is preferred to any other in the same block.

The first point above implies that the collection {B0, B1,…, Bm} is indeed a partition of C; the second implies that this partition respects the preference relation $\rightarrow$ of the query Q (assuming of course that the courses of each block are taken in the order of the blocks); and the third implies that all courses of a block can be taken in parallel. We recall that these properties are precisely those that we had required earlier on from a partition of C in order to be the answer of the preference query. More formally:

Definition 5 (Answer to a preference query): *Let Q= <S, $\rightarrow$> be a preference query, let C be the course selection based on S (i.e., the result of the initial course selection followed by completion). The answer to Q is defined to be the output of algorithm ORDPART on input C.*

Example (continued): our preference query will be evaluated as follows:

- first we apply prop.1 to the area descriptions to obtain:
  {a1}→{a3}, {a2}→{a3}, {a1,a2}→{a1},
  {a1,a2}→{a2}, {a1,a2}→{a3}, {a1,a4}→{a1},
  {a1,a4}→{a3}, {a1,a4}→{a3,a4}, {a2,a4}→{a2},
  {a2,a4}→{a3}, {a2,a4}→{a3,a4}, {a3,a4}→{a3};
- then we apply prop.2 on the set C of courses derived earlier, to arrive at the partial order of Fig.1a.
- and finally we use algorithm ORDPART to produce the ordered partition π_A which is the answer of the student's preference query:

$$[\pi_A, \rightarrow] = [\{c4, c5, c6\} \rightarrow \{c1, c2, c7\} \rightarrow \{c3\}] \qquad (13)$$

4. Combining Preferences

We now discuss the processing of queries with multiple preferences, that is queries of the form Q= <S, $\rightarrow_1$, …, $\rightarrow_k$>, where $\rightarrow_1$, …, $\rightarrow_k$ are k preference relations, one over each of k preference columns of the course catalogue. The evaluation of such queries proceeds in three steps:

1. determine the set C of courses to be taken by the student (this is done, based on the skill taxonomy and the course catalogue – see previous section);
2. process each of the following k queries Q1= <S, $\rightarrow$1>, …, Qk= <S, $\rightarrow$k> to obtain k ordered partitions π_1,…, π_κ of the set C (this is done by applying k times the ORDPART algorithm – see previous section);
3. combine the k ordered partitions π_1,…, π_κ of the set C to define the answer to Q (i.e., an ordered partition π respecting all preference relations $\rightarrow_1$, …, $\rightarrow_k$ of Q).

The first two steps are performed as explained in the previous section; therefore there is nothing new to say here. To process the third step we proceed as follows:

- define an ordering $\rightarrow$ over the set C by combining the orders of π_1,…, π_κ;
- define the ordered partition π that answers Q by applying the algorithm ORDPART with input [C, $\rightarrow$].

4.1 Defining an ordering over C

Let us first consider the set C and a single partition π_ι. We have seen that the blocks of π_ι are linearly ordered with respect to their distance (see algorithm ORDPART). For each course c in C, let oi(c) be the distance of the block of π_ι to which c belongs. In this way, each course c in C is associated to an integer oi(c) that we shall call the *i-th coordinate* of c. As a result, each course c in C is associated to a k-tuple <o1(c), …, ok(c)>, denoted by orders(c). We shall also use the following notation: orders(C)={orders(c): c∈C}

Now, assuming that we have a partial ordering $\Rightarrow$ over orders(C), we define a partial order $\rightarrow$ over C as follows:

$$c \rightarrow c' \text{ iff orders}(c) \Rightarrow \text{orders}(c'), \text{ for all c, c' in C}$$

Therefore the problem of defining a partial order over C boils down to a well known problem of order theory: given k partially ordered sets O1,…,Ok define a partial order over their Cartesian product O1×…×Ok [14]. In fact, there are several solutions to this problem, that is several partial orders that one can define over the Cartesian product. For the purposes of this paper we shall consider two such orders[4], whose definitions we recall below:

- *Lexicographic ordering*: <x1,…, xk>$\Rightarrow$<y1,…, yk> iff either x1$\rightarrow_1$ y1 or x1≡y1 and <x2,…, xk>$\Rightarrow$<y2,…, yk>
- *Pareto ordering*: <x1,…,xk>$\Rightarrow$<y1,…,yk> iff xi $\rightarrow_i$ yi, and not yj $\rightarrow_j$ xj, for some i in {1,…, k} and all j≠i.

As we can see from these two different definitions of the ordering $\Rightarrow$ over the Cartesian product, the order in which the coordinates of the two tuples, <x1,…, xk> and <y1,…, yk>, are compared plays a central role in the nature of the defined ordering. In the case of lexicographic ordering, the coordinates are compared in the order in which they appear in the tuple (which is a linear order), whereas in the case of Pareto ordering, the coordinates are compared in any order. Let us see an example of ordering the Cartesian product O1×…×Ok in our setting.

[4] Transitivity is preserved across these orders, as the partial preorder framework we employ allows us to distinguish the case of actual incomparability from the one of equally ordered items.

(1) Course	(2) π_T	(3) π_A	Distance in π Lex π_T before π_A	Distance in π Lex π_A before π_T	Distance in π Pareto over π_A, π_T
c1	0	1	1	2	1
c2	0	1	1	2	1
c3	0	2	2	4	2
c4	0	0	0	0	0
c5	1	2	3	1	1
c6	1	0	3	1	1
c7	1	1	4	3	2

Table 1. Combining preferences

Example (continued): In Table 1, the first column shows the set C of courses computed earlier. The second column shows the order of each course c in the ordered partition π_T, generated by the preference relation over period (see (1)), and the third column shows the order of c in the ordered partitions π_A, generated by the preference relation over area (see (13)). We recall that the order of c is simply the distance of the block to which c belongs. For example, c4 belongs to the first block of both π_T and π_A; c2 belongs to the first block of π_T and the second one of π_A; and so on.

Here are now some examples of ordering the courses using the ordering of tuples over π_T and π_A: if we follow lexicographic ordering with π_T before π_A then <0,2>⇨ <1,1>, therefore c3→c7; following lexicographic ordering with π_A before π_T then <1,1>⇨<0,2>, therefore c7→c3; if we follow Pareto ordering then <1,1> and <0,2> are incomparable, thus c3 and c7 are incomparable too.

4.2 Defining the ordered partition π that answers Q

Given a partial order → over the set of courses C, we can produce a linearly ordered partition π by applying the algorithm ORDPART with input [C, →].

Example (continued) Referring to Table 1, columns 4-6 summarize the results of running the algorithm ORDPART for three different priorities over the partitions π_T and π_A. Column 4 shows the distance of each course c in π under a lexicographic ordering where π_T precedes π_A; column 5 shows the distance under a lexicographic ordering where π_A precedes π_T; and column 6 shows the distance under the Pareto ordering. Based on these distances, we compute the partition under each of these three cases. We get:

lexicographic with π_T before π_A:

$$[\pi_A \cdot \pi_T, \rightarrow] = [\{c4\} \rightarrow \{c1, c2\} \rightarrow \{c3\} \rightarrow \{c5, c6\} \rightarrow \{c7\}] \quad (14)$$

lexicographic with π_A before π_T:

$$[\pi_A \cdot \pi_T, \rightarrow] = [\{c4\} \rightarrow \{c5, c6\} \rightarrow \{c1, c2\} \rightarrow \{c7\} \rightarrow \{c3\}] \quad (15)$$

pareto over π_A, π_T:

$$[\pi_A \cdot \pi_T, \rightarrow] = [\{c4\} \rightarrow \{c1, c2, c5, c6\} \rightarrow \{ c3, c7\}] \quad (16)$$

The following proposition states the properties of the partition π produced by the algorithm ORDPART.

Proposition 3 *Let* π *be the answer to the query* Q= <S, $\rightarrow_1, \ldots, \rightarrow_k$>. *Then the following hold:*

1. *in each block of* π*, other than the first block, for each course* c*, and for each preference relation* $\rightarrow_i$ *there is a course* c_i' *in the previous block such that* $c \rightarrow_i c_i'$.
2. *in each block, no course is preferred to any other course in the block under any of the preference relations* $\rightarrow_1, \ldots, \rightarrow_k$
3. $\pi = \pi_1 \cdot \ldots \cdot \pi_\kappa$ *(i.e.,* π *is the product of* $\pi_1, \ldots, \pi_\kappa$*)*[5]

The proof of this proposition follows easily from the fact that two courses c and c′ are in the same block of π iff orders(c)=orders(c′), that is iff c and c′ have the same distance in each of the partitions $\pi_1, \ldots, \pi_\kappa$.

We end this section by a remark concerning prerequisites. As the set of prerequisites satisfies reflexivity and transitivity it induces a partition of the set C of courses, in much the same way as preferences do (this partition can be computed using the algorithm ORDPART). However, the set of prerequisites is *not* a preference relation but rather a constraint imposed on curricula by the hosting institution. As a result, the partition of C induced by the prerequisites has priority over *any* preference relation, and as such it might cause "conflicts" with preference relations. However, such "conflicts" can be easily resolved by giving priority to the prerequisites partition using lexicographic prioritization. For example, we can combine the partition π_P of the prerequisite relation (17), with the partition in (14) to produce a final course sequencing π (18) in which the preference relation over period takes priority over the area one, while the prerequisite has top priority over the previous two:

$$[\pi_P, \rightarrow] = [\{c1, c3\} \rightarrow \{c2, c5\} \rightarrow \{c4, c6, c7\}] \quad (17)$$

$$[\pi, \rightarrow] = [\{c1\} \rightarrow \{c3\} \rightarrow \{c2\} \rightarrow \{c5\} \rightarrow \{c4\} \rightarrow \{c6\} \rightarrow \{c7\}] \quad (18)$$

5. Summary

We rely on a qualitative preference framework [10,11,21, 22] where query results are ordered using so-called preference relations (i.e. binary relations satisfying certain natural conditions, see [2] for a fundamental study). The main contribution of our work with respect to other qualitative frameworks lies on the use of ordered partitions of objects, generated from preference relations. Compared to [10,11] which model preferences as relations with no specific properties and to [21,22] which model preferences as strict partial orders, our approach lies somewhere in between since it treats preferences as partial preorders. It is worth finally noticing that more than one course sequence may be produced as the final outcome. For instance, one can acquire an atomic or a composite skill in different ways (e.g. the skill "Object Oriented Programming" could be acquired either by a C++ or Java course). Conceptually, we can handle such cases by considering as many different skill taxonomies as there are alternatives. However, further research is needed to make this idea concrete.

6. References

1. Antoniou, G., Baldoni, M., Baroglio, C., et al: Reasoning Methods for Personalization on the Semantic Web 2004. *Annals of Mathematics, Computing & Teleinformatics* 2,1 (2004) pp 1-24
2. Andreka H., Ryan M., Schlobbens P-Y.: Operators and Laws for Combining Preferential Relations. *Journal of Logic and Computation*, 12(1):13-53, 2002.
3. Aroyo, L., Mizoguchi, R., Tzolov, C.: OntoAIMS: Ontological Approach to Courseware Authoring. In Proc.of ICCE'03.
4. Baldoni, M., Baroglio, C., Patti, V.: Web-based Adaptive Tutoring: an Approach based on Logic Agents and Reasoning about Actions. *Artificial Intelligence Review* 22, 1 (2004) pp 3-39
5. Balke W.-T., and Güntzer U: Multi-Objective Query Processing for Database Systems. In Proc. of VLDB'04, pp 936-947.
6. Bartolini, I., Ciaccia, P., Oria, V., Özsu, T.: Flexible Integration of Multimedia Sub-queries with Qualitative Preferences. *Multimedia Tools & Applications Journal* (2006)
7. Börzsönyi S., Kossman D., Stocker K.: The Skyline Operator. In Proc. of ICDE'01
8. Brusilovsky, P.: Course Sequencing for Static Courses? Applying ITS Techniques in Large-Scale Web-Based Education. In Proc. of ITS'00
9. Brusilovsky, P.: Adaptive Hypermedia: Methods and Techniques. *International Journal of User Modeling and User-Adapted Interaction* 11 (2001) 87-110
10. Chomicki, J.: Querying with Intrinsic Preferences. In Proc. of EDBT'02 pp 34-51
11. Chomicki, J.: Preference formulas in relational queries. ACM Transactions on Database Systems (TODS), 28(4), (2003) 427-466
12. Christophides, V., Karvounarakis, G., Plexousakis D., Scholl M., Tourtounis S.: Optimizing Taxonomic Semantic Web Queries Using Labeling Schemes. In Web Semantics: Science, Services & Agents on the World Wide Web, Vol. 1(2) (2004) pp 207-228
13. Dagger, D., Wade, V., Conlan O.: Personalisation for All: Making Adaptive Course Composition Easy. *Education Technology & Society, 8* (3), pp 9-25.
14. Davey B.A., Priestly H.A.: Introduction to Lattices and Order (2nd edition). Cambridge University Press (2002)

[5] We recall that the product of two partitions π and π' is defined as $\pi \bullet \pi' = \{B \cap B' \mid B \in \pi, B' \in \pi', B \cap B' \neq \emptyset\}$, and that it is commutative and associative.

15. De Bra, P., Aerts, A., Smits, D., Stash, N.: AHA! Version 2.0, More Adaptation Flexibility for Authors. In Proc. of ELearn'2002.
16. Dolog, P., Henze, N., Nejdl, W., Sintek, M.: The Personal Reader: Personalizing and Enriching Learning Resources Using Semantic Web Technologies. In Proc. of AH'04.
17. Georgiadis, P. Foundations of Preference-based Queries. Doctoral Forum HDMS'05, Greece, 2005
18. Hafenrichter, B., Kießling, W.: Optimization of Relational Preference Queries. In Proc. of the 16th Australasian Database Conference (2005) pp 175-184
19. Henze, N., Dolog, P., Nejdl, W.: Reasoning & Ontologies for Personalized e-Learning in the Semantic Web. *Educational Technology & Society*, 7(4) (2004) pp 82-97
20. Karampiperis, P., Sampson D.: Adaptive Learning Resources Sequencing in Educational Hypermedia Systems. *Educational Technology & Society Journal*, vol. 8(4), (2005) pp 128-147
21. Kießling, W.: Foundations of Preferences in Database Systems. In Proc. VLDB'02 pp 311-322
22. Kießling, W., Köstler, G.: Preference SQL − Design, Implementation, Experiences. In Proc. of VLDB'02 pp 990-1001
23. Kravcik, M., & Specht, M.: Flexible Navigation Support in the WINDS Learning Environment for Architecture and Design. In Proc. of AH'04
24. Palazzo M. de Oliveira, J., de Freitas, V. et al: AdaptWeb: an Adaptive Web-based Courseware. In Proc. of 3rd Annual Ariadne Conference, 2003
25. Spyratos, N.: The Partition Model: A Functional Approach. INRIA Research Report No 430 (1985)
26. Spyratos, N., Christophides V.: Querying with Preferences in a Digital Library. Dagstuhl Seminar (Nº 05182) Federation over the Web LNAI 3847, 5/2005.
27. Spyratos, N.: Concepts & Algorithms for Decision Support, LRI Research Rep., 5/2006
28. Torlone, R. Ciaccia, P.: Which Are My Preferred Items? In Proc. of the W'shop on Recommendation & Personalization in e-Commerce, Spain (2002)
29. Weber, G., Brusilovsky, P.: ELM-ART: An Adaptive Versatile System for Web-based Instruction. *IJAIED Special Issue on Adaptive and Intelligent Web-Based Systems* 12(4): pp 351-384. 2001