Constraint Programming for Planning Routes in an E-learning Environment

Antonio Garrido and **Eva Onaindía** and **Oscar Sapena**
Dpto. Sistemas Informaticos y Computacion
Universidad Politecnica Valencia (Spain)
{agarridot,onaindia,osapena}@dsic.upv.es

Abstract

AI planning techniques offer very appealing possibilities for their application to e-learning environments. After all, dealing with course designs, learning routes and tasks keeps a strong resemblance with the main planning components. This paper focuses on planning learning routes under a very expressive constraint programming approach. After presenting a general planning formulation based on constraint programming, we adapt it to an e-learning setting. This requires to model learners profiles, learning concepts, how tasks attain concepts at different competence levels, synchronisation constraints for working-group tasks, capacity resource constraints and multi-criteria optimisation. Finally, we also present a simple example that shows the applicability of this model, the use of heuristics and how the resulting learning routes can be easily generated.

Introduction

Automated planning is an attractive area within AI due to its direct application to real-world problems. Actually, most everyday activities require some type of intuitive planning in terms of determining a set of tasks whose execution allows us to reach some goals under certain constraints. This direct application, the benefits it involves and, finally, the research in planning methods have made it easier the transfer of planning technology to practical applications, ranging from scientific and engineering scopes to social environments. Social environments such as education constitute an attractive field of application because of its continuous innovation and use of ICT. However, it is generally agreed that education has not yet realised the full potential of the employment of this technology. As explained in (Manouselis & Sampson 2002), this is mainly due to the fact that the traditional mode of instruction (one-to-many lecturing, or one-to-one tutoring), which is adopted in conventional education, cannot fully accommodate the different learning and studying styles, strategies and preferences of diverse learners. But now, conventional education is giving way to e-learning environments, which require learners to take the learning initiatives and control how knowledge is presented during instruction (Atolagbe 2002). Particularly, many European countries signed the Bologna joint declaration of the European space for higher education[1], which entails an important change in the learning process. With this declaration, learners roles are much more dynamic, active and autonomous. The amount of one-to-many lecturing decreases and significantly increases the amount of self-learning through the construction of coherent learning routes according to a certain instructional course design. Finally, this course design recommends sequence of educational tasks and material, tailored to individual learners needs and profiles.

In this paper we address the construction of learning routes from the viewpoint of planning based on constraint programming. After all, generating a learning route represents a planning activity with the following elements: learning goals to be attained, profile-adapted tasks with their prerequisites and learning outcomes (i.e. preconditions and effects, respectively), non-fixed durations, resources, ordering and synchronisation constraints, and collaboration/cooperation relations. The underlying idea is to plan a learning route for a learner with a given profile in order to reach some learning goals. Each route consists of a sequence of tasks, such as attending an in-person lesson, doing a lab exercise, writing a report, etc. Although intuitively each course-plan is initially created individually for each given learner, there are some particular tasks that need to be done simultaneously by several learners, such as attending a lab for the same practice work. Additionally, these tasks may require some type of synchronisation (for instance, doing a working-group task), where the start and/or end must happen at the same time. In this context, the planning component is not particularly costly since the plan is usually small (the number of tasks is between 10-20 per route, though there may be a lot of alternatives). On the contrary, the scheduling component is more significant because of the resource availability, the diversity of constraints and their handling and synchronisation among different routes. These features are not easily included in traditional planning as they require artificial mechanisms to be managed, which complicate the planning algorithms. For instance, a very frequent type of constraints in an e-learning scenario such as synchronisation

[1] Available at `http://ec.europa.eu/education/policies/educ/bologna/bologna.pdf` (accessed June 2007).

constraints, where several actions need to meet throughout a whole interval, is not easily represented and handled in planners.

Our approach for planning learning routes relies on the constraint programming formulation presented in (Garrido, Onaindía, & Arangu 2006), based on (Vidal & Geffner 2006), which encodes all type of constraints derived from both planning and scheduling features. Such a formulation provides a high level of expressiveness to deal with all the elements required in an e-learning setting and has several advantages: i) it is a purely declarative representation and, consequently, can be solved by any type of CSP solver[2]; ii) although the formulation is automatically derived from the course design, specific *ad-hoc* control information in the form of hand-coded domain knowledge or domain-dependent heuristics can be included in the formulation to make the resolution process more efficient; and iii) optimality is a major issue in this context and so different optimisation criteria can be defined *w.r.t.* the number of actions of the learning routes, the duration of their tasks or the cost associated to them. In summary, this paper introduces a formulation of planning problems by means of constraint programming and the application of such a formulation to solve a learning-route planning problem. This work is being developed under an on-going national research project whose objective is the application of AI planning techniques for the automated design, execution, monitoring and evaluation of learning routes.

This paper is organised as follows. In the second section we present the e-learning environment and its relation to AI planning, motivating some needs for using a constraint programming approach. The third section briefly reviews the formulation of a planning problem by means of constraint programming, while in the fourth section this formulation is adapted to fit an e-learning scenario. In the fifth section an example of application is analysed, showing part of the formulation, implementation and results. Finally, we present the conclusions of the paper.

E-learning and AI Planning

The application of AI planning techniques has reported important advances in the generation of automated courses within e-learning. One of the first attempts in this direction was the work in (Peachy & McCalla 1986), in which the learning material is structured in learning concepts and prerequisite knowledge is defined, which states the causal relationship between different concepts. This instructional method was one of the first approaches to combine instructional knowledge and artificial intelligence planning techniques to generate sequences of learning materials. In the same direction, Vassileva designed a system that dynamically generates instructional courses based on an explicit representation of the structure of the concepts/topics in the domain and a library of teaching materials (Vassileva 1997). Other approaches have introduced hierarchical planners to represent pedagogical objectives and tasks in order to obtain a course structure (Ullrich 2005). Most recent works, such as the one presented in (Vrakas *et al.* 2007), incorporate machine learning techniques to assist content providers in constructing learning objects that comply with the ontology concerning both learning objectives and prerequisites.

Although the task of designing learning routes for e-learning environments has been accomplished from different perspectives, one of the most intuitive approaches is the instructional planning design (van Marcke 1992), which is based on learning outcomes, information processing analysis and prerequisite analysis (Smith 2005). Therefore, e-learning can be considered as a particular planning domain with specific constraints.

The underlying idea about instructional planning is to provide a constructivist learning strategy based on both instructional tasks and instructional methods, i.e. representations of different routes to achieve the goals. An expert user, usually a teacher or instructor, designs a course as a set of learning tasks with prerequisites that are required prior to the execution of the task, learning outcomes that are attained after the execution and a positive duration. The task duration is usually a non-fixed value because in a learning environment this value cannot be precisely determined in advance as it varies among learners. Additionally, the instructor can define task-task and/or task-outcome constraints as well as deadlines to attain the learning goals. The main aim of an instructional session is to find a valid learning route for a learner to achieve the learning goals, on the basis of his/her particular constraints (e.g. personal profile, previous knowledge, resource availability and temporal constraints). In other words, an instructional design determines which learning tasks are present and in which order by using a structure of concepts/topics and tasks. As can be noted, an instructional design keeps a strong resemblance with a plan, as it is usually modelled in AI planning. Analogously, the elements necessary to find this design are similar to the elements defined traditionally in planning, since tasks can be expressed in terms of actions with duration, prerequisites and effects.

First, the course design defined by the instructor may be seen as a planning domain which contains the tasks that can be used to form the final learning routes. There is, however, slight differences in the way the e-learning domain knowledge is represented. While a planning domain is represented as a plain-text file (e.g. PDDL format (Gerevini & Long 2006)), a course design is commonly represented in a graphic way, thus explicitly modelling the structural relations between tasks. The reason for this is clear: the instructor who defines the course design neither needs being an expert in route modeling languages nor being interested in their syntax details. On the contrary, a graphic representation turns out more useful when showing the workflow among tasks (see Figure 1 below).

Second, a learning task is equivalent to an action. A task has prerequisites and learning outcomes, analogously to the conditions and effects of an action. Although a non-fixed duration is not a common feature in planning, where the du-

[2]We mean by *any type of CSP solver* a solver that supports the expressiveness of our constraint model, which includes binary and non-binary constraints. In any case, a non-binary constraint can be translated into a binary one by creating new variables and constraints, though this would increase the complexity of the model.

ration of the actions is well-known, this does not involve any difficulty in a constraint programming setting as it can be simply modelled by creating a new variable in the problem that represents such a duration (see next section for more details). Regarding the definition of constraints, there exist two basic types: task-task and task-outcome ordering constraints and/or deadlines. These types of constraints are not commonly used in planning (particularly, orderings are only due to causal link relations) but their definition within constraint programming is straightforward. As we will discuss later, other more elaborate constraints can be defined similarly and easily, which makes a constraint programming setting more appealing than traditional planning.

Third, prerequisites and learning outcomes, also known as knowledge objects or simply as concepts, can be seen as fluents that are required/achieved by tasks. In planning, fluents can represent propositional or numeric information. In the former, the domain is binary: the fluent (as a boolean proposition) is either present or not. In the latter, the domain can be defined as a real or integer domain, where the range of possible values is significantly higher than two. In an e-learning environment all concepts are numeric because no concept is entirely boolean; when a learner performs a task and achieves a concept it is not as easy as getting or not getting such a concept, but several degrees of achievement can be considered. This means that all the planning process has to reason with numeric information in order to attain concepts at a certain competence level, such as achieving a given concept in a degree greater than 5. The way to achieve these concepts involves a subtle particularity as well. In traditional planning, actions modify their numeric effects in several ways by assigning a new value, increasing, decreasing or scaling its current value. However, in an e-learning domain tasks only have increasing effects, i.e. tasks can only improve the value of a concept but never worsen it. Using exclusively increasing effects means that once a learner has attained a concept through the execution of one or several tasks, no further tasks can lessen the competence level the learner has attained; that is, the learning process is a monotonically incremental process. Some authors[3] agree that performing tasks may alter the current knowledge state (value of the concepts) of the learners and even defeat some concepts already attained in the past. In a constraint programming formulation this is possible by simply expressing such an alteration as a threat to be solved. However, we consider the fact of *forgetting* concepts, i.e. expressing limited persistence on the concepts, more appealing for a real e-learning environment. This way, we may include a concept-task temporal constraint in the form: a concept can only be used as a prerequisite for a task within 40 hours of its achievement. This type of constraint can be easily modelled using the constraint programming formulation defined below.

Finally, quality assessment for learning routes plays a similar role to the optimisation process in planning. Now, the optimisation criterion can be seen from two different perspectives. From the learner's point of view, the criterion to be optimised is the length of the learning route (plan), either in terms of number of tasks, their duration or difficulty level. It is important to note that the learning route of a learner is formed by a set of sequential tasks. In planning terminology this implies having a sequential plan per learner, where no tasks are executed simultaneously. Obviously, a learner cannot perform two tasks at the same time, but it is possible and frequent to have parallel plans for different learners. From the expert or teaching center point of view, the criterion to be optimised may be associated to the cost of the tasks, usually given in terms of the cost of the used resources. For instance, if some tasks need an expensive resource the optimisation criterion will tend to reduce the usage cost by using alternative cheaper tasks. The optimisation of resource usage and cost is more a scheduling feature than a planning one itself, but a constraint programming formulation can combine both features under the same model, which again makes a constraint programming setting an interesting approach. Additionally, a multi-criteria optimisation function that combines the two viewpoints can be easily defined to take into consideration a more representative metric.

[3]Dimitris Vrakas (2007), personal communication.

Planning as a Constraint Programming Formulation

In this section we present a general model to formulate planning as constraint programming that will be used as a basis for the e-learning scenario. Constraint programming formulations have been used in many approaches to handle both planning and scheduling features. A common feature that appears in these approaches is that they rely on constraint satisfaction techniques to represent and manage all different types of constraints, including the necessary constraints to support preconditions, mutex relations and subgoal preserving. That is, they use constraint programming for planning by encoding a planning problem as a CSP. Therefore, CSP formulations for planning include reasoning mechanisms to represent and manage the causal structure of a plan as well as constraints that denote metric, temporal and resource constraints. This general formulation through constraint programming has the ability to solve a planning problem with very elaborate models of actions. Moreover, automated formulations, like the ones presented in (Vidal & Geffner 2006; Garrido, Onaindía, & Arangu 2006) have the advantage that once the constraint programming model is formulated, they can be solved by any CSP solver.

In a constraint programming setting, a problem is represented as a set of variables, a domain of values for each variable and a set of constraints among the variables. Variables are basically used to define actions and conditions, both propositional and numeric, required by actions, along with the actions that support these conditions and the time when these conditions occur (time is modelled in $\mathbb{R}$). Variables are defined for each action present in the problem, which may comprise all actions of the planning domain (after grounding all operators), or a smaller subset. Every action a is represented by the following basic variables (Garrido, Onaindía, & Arangu 2006):

- $S(a), E(a)$ represent the start and end time of action a.

- $dur(a)$ represents the duration of action a.
- $InPlan(a)$ encodes a binary variable that denotes the presence of a in the solution plan.
- $Sup(c, a)$ represents the action that supports condition c for action a, where c represents a fluent that can be either propositional or numeric.
- $Time(c, a)$ represents the time when the causal link $Sup(c, a)$ happens; if c is a numeric condition, $Time(c, a)$ represents the time when the action in $Sup(c, a)$ last updates c.
- $Req_{start}(c, a)$ and $Req_{end}(c, a)$ represent the interval in which action a requires condition c. These variables provide a high expressiveness for representing a wide type of conditions, from punctual conditions to conditions required throughout an interval beyond the action duration.
- $V_{actual}(c, a)$ is a variable only used for numeric conditions which denotes the value of the corresponding fluent c at time $Req_{start}(c, a)$ if a requires condition c; otherwise, this variable stores the actual value of the numeric fluent at time $S(a)$.
- $V_{updated}(c, a)$ represents the new value for the fluent c updated by action a. This variable is only necessary in case a modifies the fluent c.

Constraints represent relations among variables and correspond to assignments and bindings of the variables, supporting, temporal and numeric constraints. The basic constraints defined for each variable that involves action a are:

- $S(a) + dur(a) = E(a)$ binds the variables start and end of any action a.
- $E(Start) \leq S(a)$ represents that any action a must start after fictitious action $Start$.
- $E(a) \leq S(End)$ represents that any action a must finish before fictitious action End.
- $Time(c, a) \leq Req_{start}(c, a)$ forces to satisfy condition c (either propositional or numeric) before it is required.
- $Time(c, a)$ represents the time when the action in $Sup(c, a)$ adds or updates (propositional or numeric) condition c. When c is a numeric condition, $V_{actual}(c, a) = V_{updated}(c, Sup(c, a))$.
- $Cond(c, a) = V_{actual}(c, a)$ `comp-op` $expression$, where `comp-op` $\in \{<, \leq, =, \geq, >, \neq\}$ and $expression$ is any combination of variables and/or values that is evaluated in $\mathbb{R}$, which represents the condition that the corresponding fluent c must satisfy in $[Req_{start}(c, a), Req_{end}(c, a)]$ for action a.
- Branching. $Sup(c, a) = b_i \wedge Sup(c, a) \neq b_j \mid \forall b_i, b_j$ $(b_i \neq b_j)$ that supports c for a, which represents all the possibilities to support c, one for each b_i (while $|Sup(c, a)| > 1$).
- Solving threats. Let $time\_threat(b_i)$ be the time when action b_i threatens the causal link $Sup(c, a)$, i.e. when b_i changes the value of c generated by $Sup(c, a)$. In that case, the constraint $(time\_threat(b_i) < Time(c, a)) \vee (Req_{end}(c, a) < time\_threat(b_i))$ must hold, which represents the idea of threat resolution by using promotion or demotion.
- Solving mutexes. Let $time(b_i, c)$ and $time(b_j, c)$ ($b_i \neq b_j$) be the time when b_i and b_j modify c, respectively; if c is a propositional fluent b_i/b_j generates/deletes c, whereas if c is a numeric fluent b_i and b_j give different values to c. Hence, $\forall b_i, b_j$: $time(b_i, c) \neq time(b_j, c)$ must hold, which represents the mutex resolution between the two actions being executed in parallel: b_i and b_j cannot modify c at the same time.

This flexible formulation also admits the specification of complex planning constraints such as persistence of concepts, temporal windows in the form of external constraints or general customised n-ary constraints to encode complex constraints that involve several variables of the model. Note that despite the high number of constraints in the model, they are only essential when involving actions with their variables $InPlan() = 1$.

The expressiveness of this model formulation facilitates the encoding of any planning problem, from purely propositional problems to more elaborate domains which mix propositional and numeric information, along with more complex constraints. In general, the resolution of the constraint programming model is a hard task, which becomes even more difficult when there exists many variables to be instantiated and constraints to fulfill. However, the most costly task in the overall resolution process is selecting the values for variables $Sup(c, a)$, i.e establishing the causal links of the actions that create the causal structure of the plan. On the contrary, this formulation model shows very efficient when the main aim is only to schedule plans or solve problems with medium/low load of planning. In this case, variables $InPlan()$ are already instantiated; the plan is already known and the only task is to assign the execution times of the actions and satisfy all problem constraints. In other words, this model turns out to be very appropriate in those problems where planning is not the big deal, that is, for *pseudo-planning* problems with a high load of scheduling.

Regarding the formal properties of this formulation, it inherits the properties of a POCL approach such as soundness, completeness and optimality. Soundness and completeness are guaranteed by i) the definition of the model itself, because all the alternatives to support causal links and solve threats and mutexes are considered, and ii) the completeness of the CSP solver, which performs a complete exploration of the domain of each variable. Optimality is also guaranteed by the CSP solver by performing an exhaustive, complete search until finding the best quality solution.

Adapting the Constraint Programming Formulation for E-learning

The general constraint programming formulation presented in the previous section needs to be slightly changed in order to be adapted to an e-learning environment. On the one hand, it needs to be simplified for all features related to: i) variables and constraints for propositional information, which are not used in this environment, ii) variables

Req_{start}, Req_{end} that are no longer necessary because conditions are required to be satisfied only at the beginning of each task and they are monotonically incremented, and iii) mutex-solving constraints that are now unnecessary as plans are sequential and no tasks for the same learner are executed in parallel. On the other hand, the formulation needs to be extended to include: i) constraints to guarantee a sequential plan per learner, though different learners' plans are in parallel, ii) synchronisation constraints for working-group tasks, and iii) capacity constraints to avoid overexceeding resources capacity such as labs, classrooms, etc. Now, the constraint model must include the following constraints:

- Elimination of Req_{start}, Req_{end}: the constraint $Time(c,a) \leq Req_{start}(c,a)$ now becomes $Time(c,a) \leq S(a)$.
- Sequential plan per learner: let T_{l_i} be the set of all possible tasks that a learner l_i could execute. The constraint $\forall t_j, t_k (t_j \neq t_k) \in T_{l_i} : (E(t_j) \leq S(t_k)) \vee (E(t_k) \leq S(t_j))$ must hold.
- Synchronisation of working-group tasks: let $\{t_{l_i}, t_{l_{i+1}} \dots t_{l_{i+n}}\}$ be the tasks that learners $l_i, l_{i+1} \dots l_{i+n}$ must respectively execute at the same time as a common working-group task. The constraint $(S(t_{l_i}) = S(t_{l_{i+1}}) = \dots = S(t_{l_{i+n}})) \wedge (E(t_{l_i}) = E(t_{l_{i+1}}) = \dots = E(t_{l_{i+n}}))$ must hold (obviously all the durations must be the same). Additionally, if these tasks require a particular resource R_j, such as a lab, special equipment, etc., they need to fit in the temporal window of the resource availability given by $[\min(tw(R_j)), \max(tw(R_j))]$. This way, the next constraint must also hold: $(\min(tw(R_j)) \leq S(t_{l_i})) \wedge (E(t_{l_i}) \leq \max(tw(R_j)))$.
- Resource capacity: let $T = \{t_{l_i}, t_{l_{i+1}} \dots t_{l_{i+n}}\}$ be the set of all tasks that are executed simultaneously (all the starting and ending points coincide, respectively) by different learners and require a resource R_j. Assuming that these tasks consume some quantity of a resource R_j (denoted by $use(t_{l_i}, R_j)$), the next constraint to avoid resource overconsumption must hold: $\sum_{i=1..n} use(t_{l_i}, R_j) \leq C(R_j)$, where $C(R_j)$ is the max capacity of resource R_j. This ensures that at any time throughout the execution of the tasks in T, the sum of all the individual resource consumption of the tasks does not exceed the resource capacity.

Domain-dependent heuristics

When solving a planning problem, the use of adequate heuristics becomes essential to improve the efficiency of the search and, consequently, the overall performance. Clearly, the same happens when solving a constraint satisfaction problem and especially when the problem represents a planning problem that contains many variables and constraints. One of the most effective points to apply heuristics is the branching point, i.e. when the CSP solver needs to assign a value to the variables $InPlan()$, $Sup()$ and $S()/E()$ (note that the value of the remaining variables of the model comes from a propagation of the assignment for these variables). In this point, the heuristics can be defined as the usual variable and value selection heuristics in order to reduce the branching factor, that is, which variable to select first and which value to instantiate first, respectively. Traditionally, CSP heuristics use domain-independent information to estimate this selection order, such as first select the variable with the max number of constraints, or the one with the min domain, or instantiate the values in an increasing, decreasing or random order. However, this may not be the best approach to tackle an e-learning planning problem as can be seen in the next example.

Let us assume an e-learning setting with the tasks $T_{l_1} = \{t_{1l_1}, t_{2l_1} \dots t_{il_1}\}$ and $T_{l_2} = \{t_{1l_2}, t_{2l_2} \dots t_{jl_2}\}$ that can be included in the learning routes (plans) for learners l_1 and l_2, respectively. Since the aim of the CSP solver is to find which tasks will be part of the solution, when using a *blind* heuristic the variable selection could try to instantiate first the variables associated with task t_{1l_1}, then t_{1l_2}, t_{2l_1}, t_{2l_2} and so on, i.e. alternating tasks of the two different learners in a breath-first strategy. If there appears a conflict because of the selected tasks for a learner, a lot of unnecessary backtracking will be done on the tasks of the other learner. For instance, if task t_{1l_1} was wrongly chosen, the CSP solver will need to backtrack on the already-instantiated tasks t_{1l_2}, t_{2l_2}, etc. that will not fix the problem of learner l_1 and will imply a lot of thrashing. This indication of inefficiency is much more significant when the number of learners increases and, particularly, in problems with symmetry, where a lot of effort is wasted trying to unsuccessfully instantiate almost identical tasks for different learners. Although there are some works about more efficient ways to guide backtracking, learn from conflicts and exploit symmetry when planning as a CSP (Kambhampati 2000; Zimmerman & Kambhampati 1999), we can apply a very effective domain-dependent heuristic by simply grouping the variables *w.r.t.* the learner they belong to. Thus, the selection strategy selects first the variables of one learner and does not move to a second learner until finding a valid learning route for the first one. This can be seen as a depth-first strategy, which though it does not avoid backtracking in conflicting cases with different learners sharing the same oversubscripted resource, it shows very effective in most situations. Actually, this simple strategy allows to find more learning routes for more learners in less time. Further, this heuristic has two additional advantages: i) it is valid for any CSP solver, and ii) it does not introduce an overhead in the solving process as the variable grouping can be computed before solving the problem, i.e. the grouping is independent of the solving process itself.

An E-learning Scenario of Application

In this section we present an e-learning scenario that will be used as an application example to show how to plan learning routes under a constraint programming approach. We assume that an expert defines the course design depicted in Figure 1, which consists of 7 concepts (6 + 1 previous concept) and 9 tasks of different non-fixed duration. Each concept represents a learning object, i.e. a knowledge item that can be attained from one or more tasks. Each task may rep-

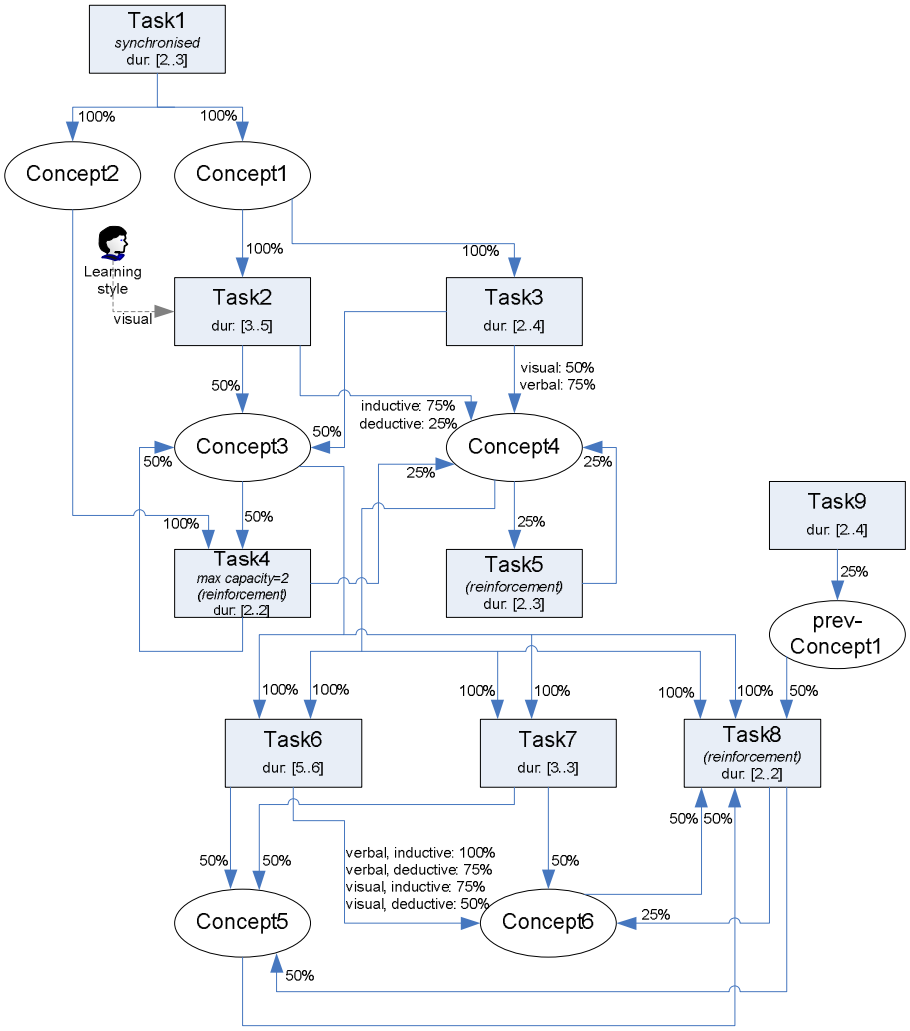

Figure 1: Course design for the e-learning scenario of application. Some tasks and concepts generated as effects are profile-dependent. The duration of the tasks is indicated between brackets. Note that tasks and concepts represent the idea of actions, preconditions and effects used in planning.

resent a discrete lesson, a seminar activity, a public talk or even a higher level course. For simplicity matters we assume that each learner can execute each task only once[4]. The course shown in Figure 1 is appropriate for learners with different learning styles, such as input profile (visual or verbal) or organisation profile (inductive or deductive), following the classification given in (Felder & Silverman 1988). According to the learner's profile, (s)he can perform, or not, a given task. Particularly, Task2 is only adequate for learners with a visual input profile. Moreover, tasks attain concepts at different competence levels (percentages) depending on the type of profile they are applied to. For instance, Task3 generates Concept4 at different levels if the learner is visual or verbal. We also assume that Task1 is an in-person lesson that needs to be performed in a synchronised way for all the learners, while Task4 requires a particular resource of max capacity 2, i.e. only two learners can perform such a task at the same time.

We apply the previous design course in an e-learning scenario with 4 learners with different profiles. Table 1 shows the profiles for these learners, the initial values of prev-Concept1 and the value required for Concept6, which is considered as the final learning goal to be attained at different competence levels.

Learner	**Profile**	**prev-Concept1**	**Concept6**
Learner1	visual, inductive	= 25	≥ 100
Learner2	verbal, inductive	= 0	≥ 75
Learner3	verbal, deductive	= 50	≥ 100
Learner4	visual, deductive	= 0	≥ 50

Table 1: Initial features of the four learners.

Formulation of the problem According to the constraint programming model presented above, the formulation for Task2 (and its concepts) of Learner1 includes the following variables and constraints:

- $InPlan(T2) \in [0, 1]$
- $S(T2), E(T2) \in [0, \infty[$
- $dur(T2) \in [3, 5]$
- $Sup(C1, T2) \in \{Start, T1\}$
- $Sup(C3, T2) \in \{Start, T3, T4, T5\}$
- $Sup(C4, T2) \in \{Start, T3, T4\}$
- $Time(C1, T2), Time(C3, T2), Time(C4, T2) \in [0, \infty[$
- $S(T2) + dur(T2) = E(T2)$
- $E(Start) \leq S(T2) < E(T2) \leq S(End)$
- $Time(C1, T2) \leq S(T2)$, $Time(C3, T2) \leq S(T2)$, $Time(C4, T2) \leq S(T2)$
- if $Sup(C1, T2) = Start$ then $Time(C1, T2) = Start$
  if $Sup(C1, T2) = T1$ then $Time(C1, T2) = T1$
- if $Sup(C3, T2) = Start$ then $Time(C3, T2) = Start$
  if $Sup(C3, T2) = T3$ then $Time(C3, T2) = E(T3)$
  if $Sup(C3, T2) = T4$ then $Time(C3, T2) = E(T4)$
  if $Sup(C3, T2) = T5$ then $Time(C3, T2) = E(T5)$
- if $Sup(C4, T2) = Start$ then $Time(C4, T2) = Start$
  if $Sup(C4, T2) = T3$ then $Time(C4, T2) = E(T3)$
  if $Sup(C4, T2) = T4$ then $Time(C4, T2) = E(T4)$
- $\forall T_i$ ($T_i \neq T2$) of Learner1: $(E(T2) \leq S(T_i)) \vee (E(T_i) \leq S(T2))$

The variables for other tasks of Learner1 and other learners are generated similarly, including the synchronisation and resource capacity constraints if necessary.

Implementation and results The constraint programming formulation is modelled and solved by Choco[5]. We have modified the variable selector of the Choco engine to take into consideration the problem variables grouped by learner, analysing first Learner1, then Learner2 and so on. The metric to optimise involves an expression with as many variables as the user requires, such as number of tasks, cost for using the labs or any combination of them. In this example we perform the optimisation task from the learner's point of view, i.e. optimising the number of actions in the four learning routes. This means to find the solution with the min number of tasks to reach the learning goals, which in this case also coincides with the routes of the shortest makespan. In such a case, Choco performs an exhaustive search of solutions until no feasible solution improves the quality of the best solution found until that time. In a problem like this, guaranteeing the optimal solution is a very expensive task, but in most cases a good solution can be found in a few seconds. Actually, Choco finds a solution very quickly and this turns to be the optimal one, though this cannot be generalised. Particularly, in this problem Choco found two solutions, the first one with 21 tasks and the second with 20, in 1 and 12 seconds respectively. Although we extended the search for more than 20 minutes no better solution was found. It is important to note that the used heuristic plays a valuable role in the solving process. For instance, we tried to solve this problem using the Choco default variable selection heuristic (first select the variable with the min domain), and the first solution was found after 12 minutes (the optimal solution took nearly 15 minutes). This shows that very simple heuristics can improve a lot the performance, with no changes in the constraint formulation at all.

The learning routes for each learner are shown in Figure 2. The longest route has a makespan of 16 (Learner1), but other learners' routes are shorter. As indicated in the problem constraints, it is important to note two properties: i) Task1 is executed at the same time by the four learners because of the synchronisation requirement of an in-person lesson, and ii) Task4 can be executed in parallel at most by two learners because of the capacity constraint.

[4]Note that when the same task can be executed more than once per learner, the constraint programming formulation needs to include a new occurrence per task (with all its variables and constraints).

[5]Choco is a Java library for constraint satisfaction problems that can be downloaded from `http://choco.sourceforge.net`

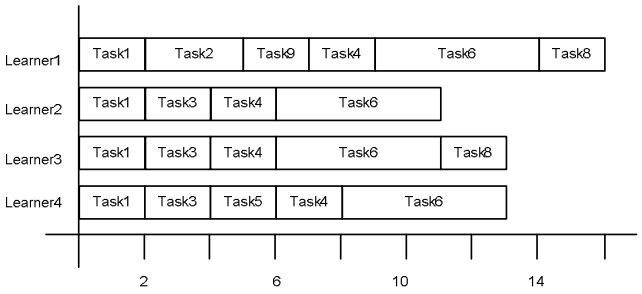

Figure 2: Resulting learning routes for each of the four learners.

Conclusions

Constraint programming formulation is a very appropriate approach to tackle planning problems that require the representation and management of a wide range of constraints, as it is the case of e-learning environments. Designing a learning route can be viewed as generating a plan in a domain where it is necessary to handle resources and the capability of such resources, synchronised tasks among several learners, orderings between tasks, deadlines or other customised and elaborate constraints. In principle, as a planning problem, the design of learning routes could be accomplished by using a current state-of-the-art planner. However, current planners cannot afford complex constraints as task synchronisation or, otherwise, it would be necessary many artificial tricks for representing and handling it. On the other hand, optimality is a major issue in e-learning contexts either from the learner or teaching center viewpoint, so it is important to guarantee good-quality plans as this might affect the overall learning route.

In an e-learning context, the activity that requires a little bit of effort is the definition of the course design by the instructor. It is necessary to classify the learning tasks, study the prerequisites and outcomes of each task (concepts), identify the profiles for which the task is best focus, determine the assessment points, establish the competence level for the concepts, etc. However, although this can be a bit tedious activity, the course design comprises the specification of a learning domain that can be used for the generation of learning routes in many different contexts (teaching centers), for different learner profiles and with different optimisation criteria. In other words, it is worth the effort devoted to course design in favor of the reusability degree we can obtain with our approach for planning routes in e-learning environments.

We can conclude by saying that the expressiveness of constraint programming makes it very appropriate for the modeling learning routes. The adequacy is also given by the fact that designing learning routes is not a very *complex* planning problem but rather a *complex* scheduling problem. For this reason, constraint programming approaches seem to be a promising direction for e-learning contexts.

Acknowledgements

This work has been partially supported by the Spanish government project MCyT TIN2005-08945-C06-06 (FEDER), by the Valencian government project GV06/096 and by the Universidad Politecnica de Valencia under the PAID-04-07 programme.

References

Atolagbe, T. 2002. E-learning: the use of components technologies and artificial intelligence for management and delivery of instruction. In *Proc. 24th Int. Conference on Information Technology Interfaces (ITI–2002)*, 121–128.

Felder, R., and Silverman, L. 1988. Learning and teaching styles in engineering education. *Engr. Education* 78(7):674–681.

Garrido, A.; Onaindía, E.; and Arangu, M. 2006. Using constraint programming to model complex plans in an integrated approach for planning and scheduling. In *Proc. 25th UK Planning and Scheduling SIG Workshop*, 137–144.

Gerevini, A., and Long, D. 2006. Plan constraints and preferences in PDDL3. In *Proc. ICAPS-2006 International Planning Competition*, 7–13.

Kambhampati, S. 2000. Planning graph as (dynamic) CSP: Exploiting EBL, DDB and other CSP techniques in Graphplan. *Journal of Artificial Intelligence Research* 12:1–34.

Manouselis, M., and Sampson, D. 2002. Dynamic knowledge route selection for personalised learning environments using multiple criteria. In *Proc. Intelligence and Technology in Educational Applications Workshop*.

Peachy, D., and McCalla, G. 1986. Using planning techniques in intelligent systems. *International Journal of Man-Machine Studies* 24:77–98.

Smith, P.L.and Ragan, T. 2005. *Instructional Design (3rd Edition)*. Wiley.

Ullrich, C. 2005. Course generation based on HTN planning. In *Proc. 13th Annual Workshop of the SIG Adaptivity and User Modeling in Interactive Systems*, 74–79.

van Marcke, K. 1992. Instructional expertise. In *Proc. 2nd Int. Conference on Intelligent Tutoring Systems, number 608*, 234–243. Springer.

Vassileva, J. 1997. Dynamic course generation. *Communication and Information Technologies* 5(2):87–102.

Vidal, V., and Geffner, H. 2006. Branching and pruning: an optimal temporal POCL planner based on constraint programming. *Artificial Intelligence* 170:298–335.

Vrakas, D.; Tsoumakas, G.; Kokkoras, F.; Bassiliades, N.; Vlahavas, I.; and Anagnostopoulos, D. 2007. PASER: a curricula synthesis system based on automated problem solving. *Int. Journal on Teaching and Case Studies, Special Issue on "Information Systems: the New Research Agenda, the Emerging Curriculum and the New Teaching Paradigm"* 1(1/2):159–170.

Zimmerman, T., and Kambhampati, S. 1999. Exploiting symmetry in the planning-graph via explanation-guided search. In *Proc. Nat. Conference on Artificial Intelligence*.