Planning and scheduling in an e-learning environment. A constraint-programming-based approach

Article *in* Engineering Applications of Artificial Intelligence · August 2008

DOI: 10.1016/j.engappai.2008.03.009 · Source: DBLP

3 authors, including:

Antonio Garrido
Universitat Politècnica de València
68 PUBLICATIONS **598** CITATIONS
SEE PROFILE

Eva Onaindia
Universitat Politècnica de València
162 PUBLICATIONS **1,260** CITATIONS
SEE PROFILE

Engineering Applications of Artificial Intelligence 21 (2008) 733–743

Contents lists available at ScienceDirect

Engineering Applications of Artificial Intelligence

journal homepage: www.elsevier.com/locate/engappai

Planning and scheduling in an e-learning environment. A constraint-programming-based approach

Antonio Garrido *, Eva Onaindia, Oscar Sapena

Sistemas Informaticos y Computacion, Universidad Politecnica de Valencia, Camino de Vera s/n, 46022 Valencia, Spain

ARTICLE INFO

Article history:
Received 29 January 2008
Accepted 1 March 2008
Available online 22 April 2008

Keywords:
Planning
Scheduling
e-Learning
Learning routes
Constraint programming
Optimisation

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 a planning process and its main components aimed at finding which tasks must be done and when. This paper focuses on planning learning routes under a very expressive constraint programming approach for planning. After presenting the 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, multi-criteria optimisation, breaking symmetry problems and designing particular heuristics. Finally, we also present a simple example (modelled by means of an authoring tool that we are currently implementing) which shows the applicability of this model, the use of different optimisation metrics, heuristics and how the resulting learning routes can be easily generated.

1. 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 reports and, finally, the advances on the research in AI planning have facilitated the transfer of planning technology to practical applications, ranging from scientific and engineering scopes to social environments. Particularly, social environments such as education constitute an attractive field of application because of its continuous innovation and use of ICT (Information and Communication Technologies). However, it is generally agreed that education has not yet realised the full potential of the utilisation of this technology. As explained in Manouselis and 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), which is not a simple task. 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, learner's 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 automated 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, indicating *which* tasks must be done, for a learner with a given profile in order to reach some learning goals. The general scheme is shown in Fig. 1. First, an instructional course design is defined

* Corresponding author. Tel.: +34 963877007; fax: +34 963877359.
E-mail addresses: agarridot@dsic.upv.es (A. Garrido), onaindia@dsic.upv.es (E. Onaindia), osapena@dsic.upv.es (O. Sapena).

[1] Available at http://ec.europa.eu/education/policies/educ/bologna/bologna.pdf (accessed January 2008).

0952-1976/$ - see front matter © 2008 Elsevier Ltd. All rights reserved.
doi:10.1016/j.engappai.2008.03.009

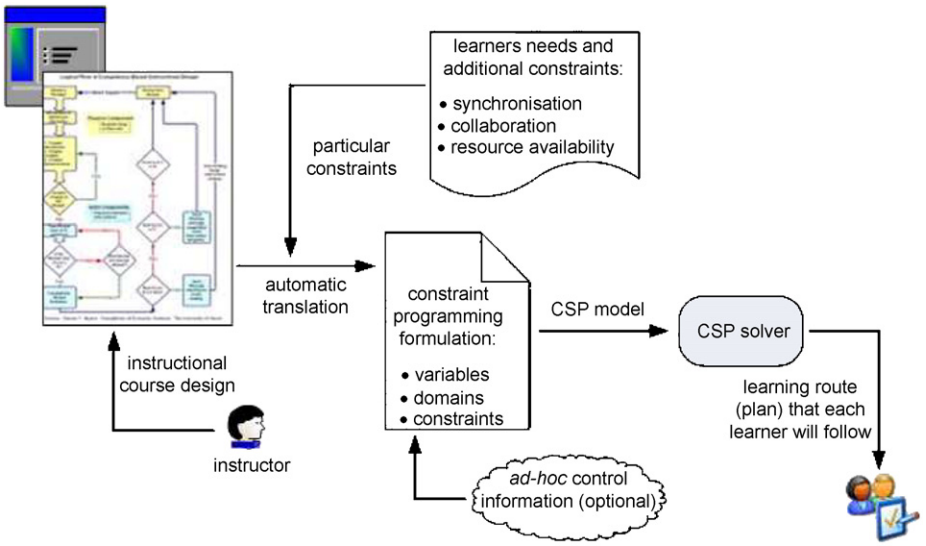

Fig. 1. General scheme for planning individual learning routes.

by means of a visual authoring tool. This course is extended with the additional, particular constraints of each application context and, all together, is translated into a constraint programming model that is later solved by a CSP solver. Finally, the output solution provides a learning route per learner, that is a profile-adapted plan. Each individual 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. Thus, a learning route also requires the time allocation of its tasks according to the temporal + resource constraints, i.e. *when* the tasks will be done (scheduling component), which are encoded as additional constraints. In this context, the planning component is not particularly costly since the plan is usually small; the number of tasks is between 10 and 20 per route, though there may be a lot of different 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 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 et al. (2006), previously based on Vidal and 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 it has several advantages:

- It is a purely declarative representation and, consequently, it can be solved by *any type* of CSP solver. Obviously, by *any type* of CSP solver we mean 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 (Bacchus and van Beek, 1998), though this would increase the complexity of the model.
- The whole formulation is automatically derived from the course design, without any need of specific hand-coded domain knowledge. This means that no expert users are required to develop our constraint model as it is straightforwardly generated from the tasks and relations provided in the instructional course. Despite this, specific *ad hoc* control information in the form of hand-coded domain knowledge or domain-dependent heuristics can be easily included in the formulation to find a better plan or make the resolution process more efficient.
- Formal properties, such as soundness, completeness and optimality, hold in our constraint model. In particular, optimality is a major issue in this context and so different optimisation multi-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 paper is organised as follows. In the second section we present some basic background on e-learning environments and their 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, which imposes some particular requirements. 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 and point some directions for future work.

2. 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 and 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 (1997) 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. 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 and Ragan, 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. As depicted in Fig. 1, 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 for this design are similar to the elements traditionally defined 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, as indicated in Table 1. While a planning domain is represented as a plain-text file (e.g. PDDL format, Gerevini and 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 to be an expert in route modelling languages nor needs to be interested in their syntax details. On the contrary, a graphic representation turns out to be more useful when showing the workflow among tasks (see Fig. 4).

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 duration 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.

Table 1
Main differences (and similarities) between e-learning and planning settings

	e-Learning	Planning
Knowledge representation and definition	Graphic workflow	Plain-text file (PDDL)
Structural elements	Learning tasks (prerequisites and outcomes)	Actions (preconditions and effects)
Duration of tasks	Non-fixed	Fixed (non-fixed is unusual)
General type of constraints	Task– task and task– outcome orderings	Causal-links
Other constraints	Deadlines, synchronisation and collaboration and/or cooperation relations	Uncommon (time initial literals)
Type of effects	Monotonically increasing (incremental process)	Any type ($+=, -=, *=, /=$ and $:=$)
Domain of variables	Numeric (marks, evaluations, etc.)	Numeric planning (more difficult than traditional planning)
Type of plan	Sequential (per learner) and parallel (for multiple learners)	Sequential or parallel
Quality assessment	Resource usage/cost, number of actions or any multi-criteria combination	Cost of the actions or, usually, planning metrics

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 just as simple as to achieve or not achieve such a concept, but several levels of achievement can be considered. This means that 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 at a level 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[2]

[2] Vrakas et al. (2007), personal communication.

agree that performing tasks may alter the current knowledge state (value of the concepts) of the learners, diminishing and even invalidating 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, for modelling this issue, we consider that implementing the idea of *forgetting* concepts, i.e. expressing limited persistence on concepts, is more suitable and 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 h 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 usually frequent, to have parallel plans for different learners. From the expert or teaching centre 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 then the optimisation criterion will tend to reduce the usage cost by selecting 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. 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 (mutual exclusion) 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 and Geffner (2006) and Garrido et al. (2006) have the advantage that once the constraint programming model is formulated, they can be solved by any type of technique implemented in the 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. A feasible solution to this problem assigns to each variable a value from its domain that satisfies all the problem constraints. In our model, 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 et al., 2006):

- $S(a), E(a)$ represent the start and end time of action a.
- $\mathrm{dur}(a)$ represents the duration of action a.
- $\mathrm{InPlan}(a)$ encodes a binary variable that denotes the presence of a in the solution plan.
- $\mathrm{Sup}(c,a)$ represents the action that supports condition c for action a, where c represents a fluent that can be either propositional or numeric.
- $\mathrm{Time}(c,a)$ represents the time when the causal link $\mathrm{Sup}(c,a)$ happens; if c is a numeric condition, $\mathrm{Time}(c,a)$ represents the time when the action in $\mathrm{Sup}(c,a)$ last updates c.
- $\mathrm{Req}_{\mathrm{start}}(c,a)$ and $\mathrm{Req}_{\mathrm{end}}(c,a)$ 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 when $\mathrm{Req}_{\mathrm{start}}(c,a)=\mathrm{Req}_{\mathrm{end}}(c,a)$ to conditions required throughout an interval beyond the action duration, for instance when $\mathrm{Req}_{\mathrm{end}}(c,a)=E(a)+5$.
- $V_{\mathrm{actual}}(c,a)$ is a variable only used for numeric conditions which denotes the value of the corresponding fluent c at time $\mathrm{Req}_{\mathrm{start}}(c,a)$ if a requires condition c; otherwise, this variable stores the actual value of the numeric fluent at time $S(a)$.
- $V_{\mathrm{updated}}(c,a)$ represents the new value for the fluent c updated by action a. This variable is only necessary in case a modifies the fluent c.

Example (*Planning variables*). Let us consider the actions shown in Fig. 2a, where the (propositional + numeric) preconditions to be satisfied before the action starts are shown in the up-left angle of the action and the effects that are generated when the action ends are shown in the down-right angle. If we focus on actions $a1$, $a3$ and $a5$, in addition to the basic variables $S()$, $E()$ and $\mathrm{InPlan}()$, we have: $\mathrm{Sup}(c4,a3)=\{a4\}$, $\mathrm{Sup}(nc3,a5)=\{a2,a3,a6\}$, $\mathrm{Sup}(c4,a5)=\{a4\}$; $\mathrm{Time}(c4,a3)=\{E(a4)\}$, $\mathrm{Time}(nc3,a5)=\{E(a2),E(a3),E(a6)\}$, $\mathrm{Time}(c4,a5)=\{E(a4)\}$ because all effects are generated in the end of each action; $\mathrm{Req}_{\mathrm{start}}(c4,a3)=\{S(a3)\}$, $\mathrm{Req}_{\mathrm{end}}(c4,a3)=\{E(a3)\}$, $\mathrm{Req}_{\mathrm{start}}(nc3,a5)=\{S(a5)\}$, $\mathrm{Req}_{\mathrm{end}}(nc3,a5)=\{E(a5)\}$, $\mathrm{Req}_{\mathrm{start}}(c4,a5)=\{S(a5)\}$, $\mathrm{Req}_{\mathrm{end}}(c4,a5)=\{E(a5)\}$ if we assume that the preconditions must hold during the whole execution of each action; $V_{\mathrm{actual}}(nc3,a3)=\{\mathrm{initial\_value}(nc3), V_{\mathrm{updated}}(nc3,a2), V_{\mathrm{updated}}(nc3,a6)\}$, $V_{\mathrm{actual}}(nc3,a5)=\{\mathrm{initial\_value}(nc3), V_{\mathrm{updated}}(nc3,a2), V_{\mathrm{updated}}(nc3,a3), V_{\mathrm{updated}}(nc3,a6)\}$ as $nc3$ has obviously an initial value and it can be also modified by $a2$, $a3$ and $a6$; $V_{\mathrm{updated}}(nc3,a3)=\{V_{\mathrm{actual}}(nc3,a3)-5\}$ as $a3$ consumes five units of $nc3$.

On the other hand, if we focus again on actions $a1$, $a3$ and $a5$ in the plan shown in Fig. 2b, the values for their previously unbound variables are finally: $\mathrm{InPlan}(a1)=\mathrm{InPlan}(a5)=1$ and $\mathrm{InPlan}(a3)=0$; $S(a1)=0$, $E(a1)=2$, $S(a5)=4$, $S(a5)=5$; $\mathrm{Sup}(nc3,a5)=a2$; $\mathrm{Time}(nc3,a5)=E(a2)$; $V_{\mathrm{actual}}(nc3,a5)=V_{\mathrm{updated}}(nc3,a2)$. Clearly, as $\mathrm{InPlan}(a3)=0$ in Fig. 2b the values of their remaining variables are not meaningful.

Constraints in our model represent relations among variables and correspond to assignments and bindings of the variables, supporting, temporal and numeric constraints. The

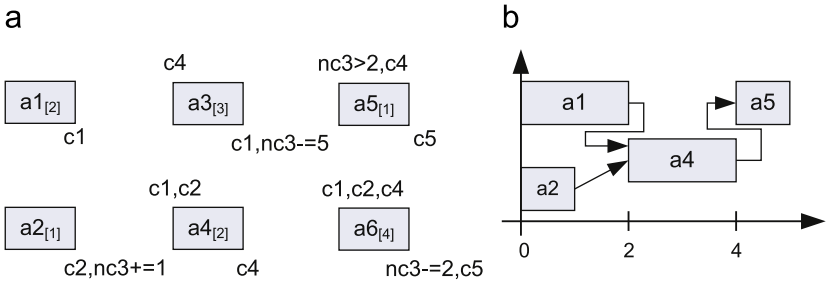

Fig. 2. Example actions. (a) Shows the actions with their preconditions (up-left angle) and effects (down-right angle); *ci* stands for propositional condition, whereas *nci* stands for numeric condition. The duration of the action is shown between brackets. (b) Shows a solution plan to find the goal $c5$. Note that replacing $a5$ by $a6$ would be also a feasible solution for achieving $c5$, with the same number of actions but longer *w.r.t.* the length (makespan) of the plan.

basic constraints defined for each variable that involves action a are:

- $S(a) + \text{dur}(a) = E(a)$ binds the variables start and end of any action a.
- $E(\text{Start}) \leqslant S(a)$ represents that any action a must start after fictitious action *Start*.[3]
- $E(a) \leqslant S(\text{End})$ represents that any action a must finish before fictitious action *End*.
- $\text{Time}(c, a)$ represents the time when the action in $\text{Sup}(c, a)$ adds or updates (propositional or numeric) condition c. When c is a numeric condition, $V_{\text{actual}}(c, a) = V_{\text{updated}}(c, \text{Sup}(c, a))$. Additionally, $\text{Time}(c, a) \leqslant \text{Req}_{\text{start}}(c, a)$ forces to satisfy condition c (either propositional or numeric) before it is required.
- $\text{Cond}(c, a) = V_{\text{actual}}(c, a)$ `comp-op` *expression*, where `comp-op` $\in \{<, \leqslant, =, \geqslant, >, \neq\}$ 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 $[\text{Req}_{\text{start}}(c, a), \text{Req}_{\text{end}}(c, a)]$ for action a. For instance, in Fig. 2a $\text{Cond}(nc3, a5) = V_{\text{actual}}(nc3, a5) > 2$.
- Branching. $|\text{Sup}(c, a) = b_i \wedge \text{Sup}(c, a) \neq b_j| \ \forall b_i, b_j \ \ (b_i \neq b_j)$ that supports c for a, which represents all the possibilities to support c, one for each b_i (while $|\text{Sup}(c, a)| > 1$).
- Solving threats. Let $\text{time\_threat}(b_i)$ be the time when action b_i threatens the causal link $\text{Sup}(c, a)$, i.e. when b_i changes the value of c generated by $\text{Sup}(c, a)$. In that case, the constraint $(\text{time\_threat}(b_i) < \text{Time}(c, a)) \vee (\text{Req}_{\text{end}}(c, a) < \text{time\_threat}(b_i))$ must hold, which represents the idea of threat resolution by using promotion or demotion.
- Solving mutexes. Let $\text{time}(b_i, c)$ and $\text{time}(b_j, c)$ $(b_i \neq b_j)$ 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, $\forall b_i, b_j$: $\text{time}(b_i, c) \neq \text{time}(b_j, c)$ 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 (Garrido et al., 2006). Note that despite the high number of variables and constraints in the model, they are only essential when involving actions with their variables $\text{InPlan}() = 1$. This means that when $\text{InPlan}(a) = 1$ all variables and constraints that action a involves need to be activated, and deactivated otherwise like in conditional/dynamic CSPs (Mittal and Falkenhainer, 1990) that support conditional activation of variables and constraints.

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 $\text{Sup}(c, a)$, 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, i.e. the actions that form the plan are already known, and the only task is to assign the execution times of the actions to 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.

3.1. Use of heuristics

The success of planning highly relies on the heuristics used to reduce the search space, which may be huge even in small problems. A constraint programming solving process can adopt a similar solution: devise heuristic estimations, automatically extracted from the problem definition, and include them in the constraint formulation, as lower and upper bounds to reduce the variables domain, thus avoiding unfruitful search. The underlying idea is to use estimations similar to the ones used in heuristic planning to approximate the cost/time to support conditions or to start actions from an initial state. The heuristic estimations can be calculated in different ways such as using relaxed planning graphs, relaxed plans or approximated (Bonet and Geffner, 2001; Do and Kambhampati, 2001; Hoffmann and Nebel, 2001; Haslum, 2006). In particular, a straightforward heuristic calculated through a relaxed planning graph can estimate the earliest time when an action can start as an optimistic measure of reachability; that is, the distance between the dummy action *Start* and each action a, namely $\delta(\text{Start}, a)$. This way, the binding constraint $E(\text{Start}) \leqslant S(a)$ now becomes $E(\text{Start}) + \delta(\text{Start}, a) \leqslant S(a)$, which approximates significantly better the start time of action a and helps reduce the search space.

In addition to the definition and use of heuristic estimations, other estimations can also be used in the CSP solver as branching heuristics, i.e. heuristics to be applied at the branching point that appears when supporting a causal link $\text{Sup}(c, a)$. These heuristics can be defined as the usual variable or value selection heuristics. In the former, the variable to be selected first can be that one with

[3] As commonly used in planning, we include two dummy actions *Start* and *End* that represent the first and last action of the plan, respectively.

the max number of constraints, or that one with the min domain to help reduce the branching factor. In the latter, the heuristic can provide an ordering criterion for the application of the actions $\{a_i\}$ that support such a causal link (value ordering). For instance, the heuristic estimation may determine to select first action a_i with min value $\delta(\text{Start}, a_i)$, i.e. the action that first appears, or simply the latest to reduce the slack. It is obvious that these heuristic estimations are very valuable in a wide variety of problems with many variables and constraints. Furthermore, we can apply other techniques to the CSP engine that can also improve the performance of the solving process, such as forward checking, arc-consistency or lookahead techniques.

An important advantage is that the application of heuristic strategies as pruning mechanisms improves the overall solving process thanks to the use of bounds that reduce the variables domain and help discard some unfeasible partial solutions. Moreover, heuristic calculations, particularly those which are included in the problem formulation, do not introduce an overhead in the solving process as they can be computed in a preprocessing stage before solving the problem, making the heuristic computation independent from the solving process itself.

3.2. Formal properties

This constraint programming formulation has a strong resemblance with a POCL (partial-order causal-link) approach by combining partial-order planning and least-commitment for reasoning on causal links and threat resolution (Penberthy and Weld, 1992; Vidal and Geffner, 2006; Weld, 1994). Consequently, this formulation shares the same formal properties as a POCL approach such as soundness, completeness and optimality, making it an appealing framework for planning. Nevertheless, in order to guarantee these nice properties during the CSP resolution some support on the CSP solver side becomes necessary as well. First of all, soundness and completeness can be easily guaranteed by (i) the definition of the model itself (this factor is independent of the CSP solver), because all the alternatives to support causal links and solve threats and mutexes are considered and (ii) the completeness of the CSP solver. This guarantees that if there exists a solution it will be found by the solver. Second, optimality can be also guaranteed when the CSP solver performs an exhaustive, complete search until finding the best quality solution. Obviously, optimality also depends on the granularity supported by the CSP solver. Let us assume a planning problem with only two actions a and a' with duration 1. Clearly, if the model imposes an ordering constraint like $S(a) < S(a')$ and $S(a) = 1$, then $S(a') = 2$ is an optimal solution if the CSP solver does not support continuous domains, i.e. if all the domains of variables are discrete. But $S(a') = 1.1$ or $S(a') = 1.01$ would be also optimal solutions in higher precision solvers. This means that all the '$<$' or '$>$' constraints are internally managed by the solver like '$\leqslant$' or '$\geqslant$', respectively, plus the corresponding value of the solver's precision. Although this may sound a bit odd, it is not particularly relevant for the constraint programming formulation as plan validation and evaluation requires as input the minimal precision with which we evaluate the quality of the plans (Long and Fox, 2001).

All in all, we can conclude that this formulation provides a general and property-compliant model that can be used by many CSP solvers and if the CSP solver performs a complete search then the solution will not be only sound but also optimal.

4. Adapting the constraint programming formulation for e-learning

As indicated in Section 2, e-learning can be seen as a particular planning problem which, consequently, can be encoded as a constraint programming formulation. However, 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, the formulation model can be simplified because:

- Variables and constraints for propositional information are not used in this environment, because no concept is entirely boolean. In consequence, all variables and constraints in the model involve numeric conditions now.
- Variables $\text{Req}_{\text{start}}$, Req_{end} are no longer needed; the high level of expressiveness that these two variables provide is unnecessary because conditions are required to be satisfied only at the beginning of each task and they are monotonically incremented.
- Mutex-solving constraints are now unnecessary as plans are sequential (per learner) and no tasks for the same learner are executed in parallel.

These simplifications imply a reduction in the model complexity. But, on the other hand, the formulation needs to be extended to include some new additional requirements:

- Extra constraints to guarantee a sequential plan per learner. Although different learners' plans run in parallel, no actions for the same learner are allowed concurrently.
- Synchronisation constraints for working-group tasks. In an e-learning setting some tasks among several learners need to be executed simultaneously and at a particular time (e.g. attending an in-person seminar activity that must fit in a lab timetable).
- Capacity constraints to avoid overexceeding resources capacity such as labs, classrooms, scarce devices, etc.

These extra constraints impose some new complexity on the model, but on the other hand they help reduce the variable domains, which in general makes the problem solving process easier.

More formally, all the previous changes are encoded in the constraint formulation in the following way:

- Elimination of $\text{Req}_{\text{start}}$, Req_{end}. This way, the constraint $\text{Time}(c, a) \leqslant \text{Req}_{\text{start}}(c, a)$ now becomes $\text{Time}(c, a) \leqslant S(a)$.
- Sequential plan per learner. Let T_{l_i} be the set of all possible tasks that a learner l_i could execute. The constraint $\forall t_j, t_k (t_j \neq t_k) \in T_{l_i} : (E(t_j) \leqslant S(t_k)) \vee (E(t_k) \leqslant S(t_j))$ must hold.
- Synchronisation of working-group tasks. Let $\{t_{l_i}, t_{l_{i+1}}, \ldots, t_{l_{i+n}}\}$ be the tasks that learners $l_i, l_{i+1}, \ldots, l_{i+n}$ must, respectively, execute at the same time as a common working-group task. The constraint $(S(t_{l_i}) = S(t_{l_{i+1}}) = \cdots = S(t_{l_{i+n}})) \wedge (E(t_{l_i}) = E(t_{l_{i+1}}) = \cdots = E(t_{l_{i+n}}))$ 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 $[\min(tw(R_j)), \max(tw(R_j))]$. Thus, the next constraint must also hold: $(\min(tw(R_j)) \leqslant S(t_{l_i})) \wedge (E(t_{l_i}) \leqslant \max(tw(R_j)))$.
- Resource capacity. Let $T = \{t_{l_i}, t_{l_{i+1}}, \ldots, t_{l_{i+n}}\}$ be the set of all tasks that are executed concurrently (the starting and ending points, respectively, may coincide or not) by different learners and require a given resource R_j. Assuming that these tasks consume some quantity of resource R_j (denoted by $\text{use}(t_{l_i}, R_j)$), the next constraint to avoid resource overconsumption must hold: $\sum_{i=1,\ldots,n} \text{use}(t_{l_i}, R_j) \leqslant C(R_j)$, where $C(R_j)$ is the max capacity of resource R_j. This ensures that, at any time throughout the execution of tasks in T, the sum of all the individual resource

consumption of the tasks does not exceed the resource capacity.

4.1. Repeating tasks. Breaking symmetry

In an e-learning setting it may be necessary to have tasks that can be executed more than once in order to achieve a particular knowledge level. This is the situation in which a task acts as a reinforcement task (see Fig. 3), i.e. a task that a learner may need to execute several times, or simply a task that needs to be repeated when the learner fails to achieve its learning outcomes. This involves a high degree of flexibility in the education scenario, as the precise number of repetitions is chosen by the CSP solver according to the learner's necessities or preferences. However, this number of repetitions is not unlimited as there usually exist educational restrictions that prevent the learner from doing the same task an infinite number of times.

In our constraint programming formulation, repeating tasks are modelled as other tasks with the only difference that one independent instance of each task is needed per repetition; actually, this means having different occurrences at different times of the same task. Hence, if a task can be repeated at most three times, three tasks (t_{rep1}, t_{rep2} and t_{rep3}) must be included in the model. This is necessary to simulate the fact of having one, two or the three repetitions in the solution plan. Initially, all the instances of the same task are *clones* that have the same domain and the same constraints. Later, the CSP solver may decide to include one or more of them in the plan, making $\text{InPlan}() = 1$ and keeping the others untouched. The main drawback here is that this can entail a symmetry problem.[4] During the solving process, any instance in $\{t_{\text{rep1}}, t_{\text{rep2}}, t_{\text{rep3}}\}$ can be planned, and in any order. Particularly, if the CSP solver selects first t_{rep1} and this decision leads to an unfeasible solution, the solver needs to backtrack. After this backtracking, the solver would select any instance in $\{t_{\text{rep2}}, t_{\text{rep3}}\}$, leading again to two unfeasible solutions, one after selecting each t_{rep2} and t_{rep3}, respectively. Loosely speaking, in such a situation the CSP solver wastes a lot of effort trying to plan, unsuccessfully, identical tasks. This is a clear inefficiency that may lead to redundant search and significantly diminish the overall performance of the search.

Fortunately, we can extend the constraint formulation to break this symmetry in an easy way. Let t be a task that can be repeated at most n times. This makes necessary the generation of $\{t_{\text{rep1}}, t_{\text{rep2}}, \ldots, t_{\text{repn}}\}$ instances in the model, with all their variables and constraints. Additionally, the following constraints must be also included:

for$(i = 1; i < n; i++)$

if $\text{InPlan}(t_{\text{repi}}) = 0$ then $\text{InPlan}(t_{\text{repi}+1}) = 0$.

The purpose of this loop aims at generating constraints of the type 'if $\text{InPlan}(t_{\text{rep1}}) = 0$ then $\text{InPlan}(t_{\text{rep2}}) = 0$', 'if $\text{InPlan}(t_{\text{rep2}}) = 0$ then $\text{InPlan}(t_{\text{rep3}}) = 0$' and so on. Intuitively, these constraints force the CSP solver to plan the tasks $\text{InPlan}(t_{\text{repi}})$ in order. For instance, the task t_{rep3} will not be planned unless both t_{rep1} and t_{rep2} are in the plan, which prevents the search from analysing symmetric situations, thus avoiding the inefficiency pointed above.

4.2. 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 at 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 these variables). At this point, the heuristics that can be defined are the usual variable and value selection heuristics aimed at reducing 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 establish this selection order, such as first select the variable with the max number of constraints involved, 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.

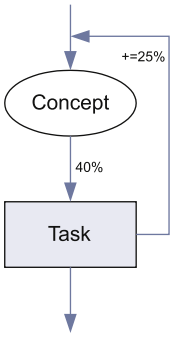

Fig. 3. A reinforcement task that could be executed several times. The task requires some level of knowledge of a concept, supports the same concept (and possibly others) and can be executed more than once. Generally, the instructor also needs to specify the max number of repetitions to avoid *infinite loops*.

Let us assume an e-learning setting with the tasks $T_{l_1} = \{t_{1l_1}, t_{2l_1}, \ldots, t_{il_1}\}$ and $T_{l_2} = \{t_{1l_2}, t_{2l_2}, \ldots, t_{jl_2}\}$ that can be included in the learning routes (plans) for learners l_1 and l_2, respectively. Since the goal of the CSP solver is to find which tasks will be part of the solution, a *blind* variable selection heuristic will 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 learners in a breadth first fashion. If there appears a conflict because of the selected tasks for a learner then a lot of unnecessary backtracking will be performed on the tasks of the other learner. For instance, if task t_{1l_1} is wrongly chosen then the CSP solver will need to backtrack on the already-instantiated tasks t_{1l_2}, t_{2l_2}, etc. This will never fix the problem of the wrong selection for learner l_1 in the allocation of task t_{1l_1} and, additionally, it will entail a lot of thrashing. This indication of inefficiency is much more significant when the number of learners increases and, particularly, when they need to do almost identical tasks. Although there are some works in domain-independent planning about more efficient ways to guide backtracking, learn from conflicts and exploit symmetry when planning as a CSP (Kambhampati, 2000; Zimmerman and Kambhampati, 1999), we can apply a very effective domain-dependent heuristic for our e-learning scenario by simply grouping together the variables related to the same learner. Thus, the selection strategy selects first the variables of one learner and does not proceed with a second learner until finding a valid learning route for the first one. This depth-first-like exploration turns out to be very effective in most situations. However, it must be noticed that it does not avoid backtracking in conflicting cases where different learners share the same oversubscripted resource. Actually, this simple strategy allows finding more learning routes

[4] We thank Roman Bartak for pointing out this drawback.

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.

5. An e-learning scenario of application

In this section we present a simple e-learning scenario that will be used as an application example to show how to plan learning routes under a constraint programming approach. As part of an on-going national research project, we are implementing an authoring tool in Visual C# to help design the e-learning course in a simple way (see Fig. 4). The underlying idea of this tool is to have a user-friendly graphical application with intuitive input forms, as shown in Fig. 5. These forms provide instructors support during the specification of a course by means of the definition of learners profiles, concepts, tasks and their relations. For our application example, we assume that an expert defines the course design depicted in Fig. 4, which consists of seven concepts ($6+1$ previous concept) and nine tasks of different non-fixed duration. Note that Task5 can be executed up to two times, so actually the model contains 10 tasks, including $\text{Task5}_{\text{rep1}}$ and $\text{Task5}_{\text{rep2}}$. Each concept represents a learning object, i.e. a knowledge item that can be attained from one or more tasks. Each task may represent a discrete lesson, a seminar activity, a public talk or even a high-level course. Each learner can execute each task only once, with the exception of Task5 that can be executed at most twice as indicated in Fig. 5 by its value max repetitions $= 2$.

The course shown in Fig. 4 is appropriate for learners with different learning styles, given by the input profile (visual or verbal) or the organisation profile (inductive or deductive), following the classification given in Felder and Silverman (1988). According to these profiles, learners 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 suitable for. For instance, Task3 generates Concept4 at different levels: if the learner is visual the task increases the value of Concept4 in 50%, whereas it increases such a value in 75% if the learner is 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 simultaneously.

We apply the previous design course in an e-learning scenario with four learners with different profiles. Table 2 shows the profiles for these learners, the initial values of prev-Concept1 and the minimum competence levels required for Concept6, which, in

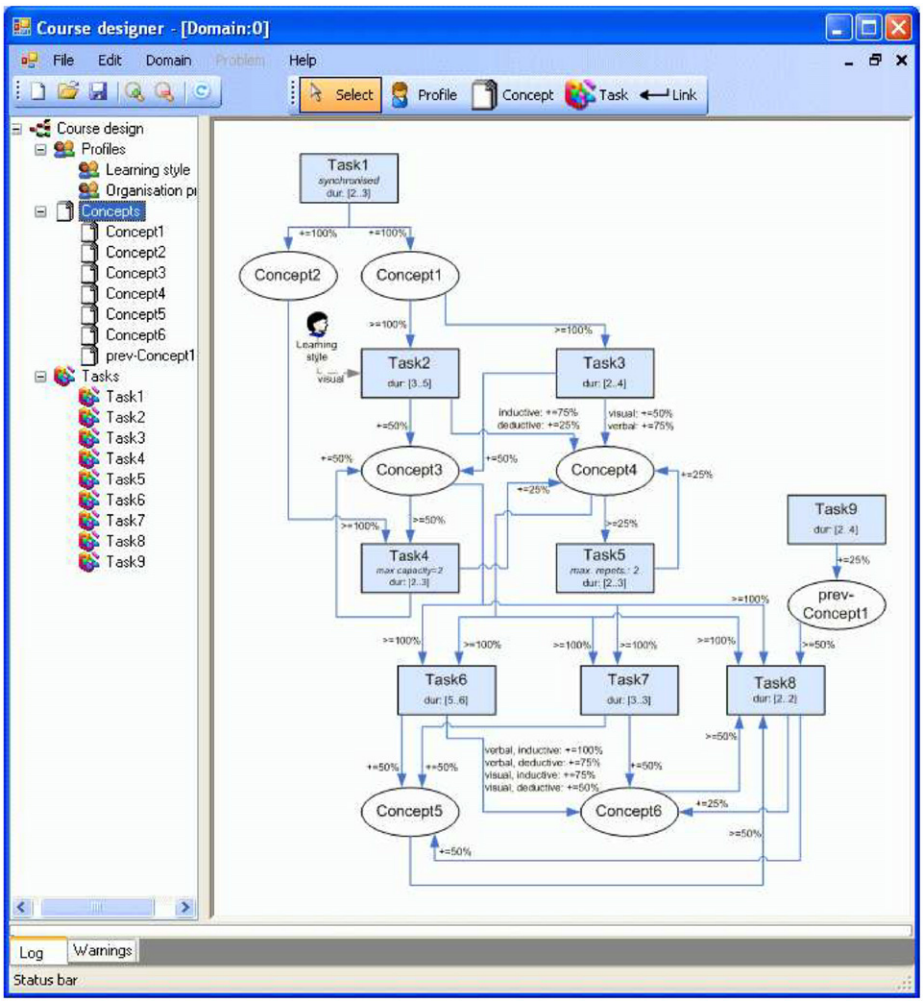

Fig. 4. Snapshot of the prototype for the authoring tool with the course design for the e-learning scenario of application. Some tasks, such as Task2 or Task7, and concepts generated as effects, such as Concept4 or Concept6, are profile-dependent. The duration of the tasks is indicated between brackets. Note that tasks and concepts represent the idea of actions, preconditions and effects used in planning.

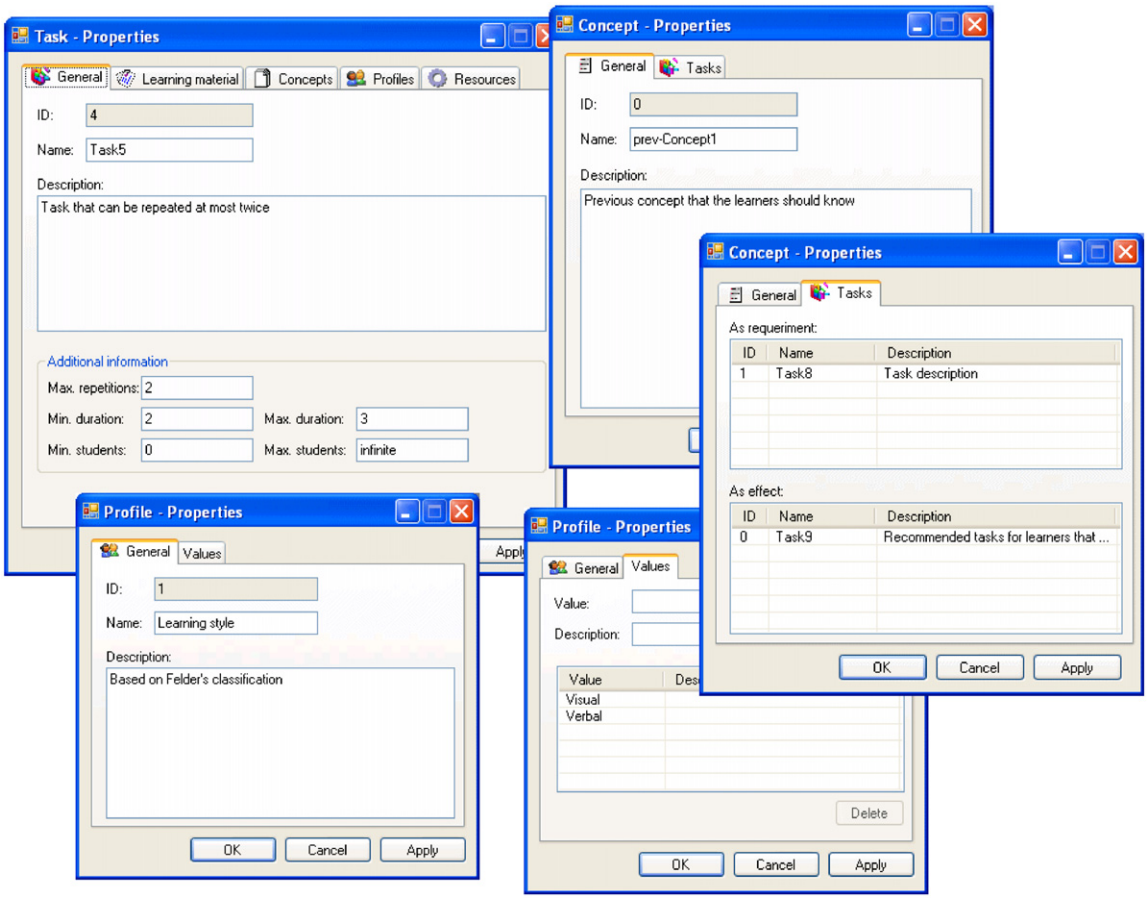

Fig. 5. Some input forms of the authoring tool to define the properties of tasks, concepts and learners profiles. In addition to this information, essential for planning e-learning routes, the tool also allows to define learning material (slides, multimedia documents, handouts, etc.), the types they belong to (visual, multimedia, Powerpoint presentation, etc.) and the resources (computers, books, etc.) that they required.

Table 2
Initial features of the four learners that represent a particular problem

Learner	Profile	prev-Concept1	Concept6
Learner1	Visual, inductive	$=25$	$\geqslant 100$
Learner2	Verbal, inductive	$=0$	$\geqslant 75$
Learner3	Verbal, deductive	$=50$	$\geqslant 100$
Learner4	Visual, deductive	$=0$	$\geqslant 50$

The initial value for the remaining concepts are initially 0.

this example, is the final learning goal to be attained. Note that the course design represents a general domain of application that can be used in many different scenarios (particular problems). This way, the same course can be used in another scenario with other learners, different profiles and with other learning goals and/or competence levels.

5.1. 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:

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

The variables for other tasks of Learner1 and other learners are generated similarly, including the synchronisation constraint for Task1 and the resource capacity constraints for Task4. In addition to this, it is also necessary to define the metric function. The metric to optimise may involve an expression with as many variables as the user requires, such as makespan (overall duration of the plan), number of tasks, cost of lab usage or any combination of them. In our example we perform the optimisation task from the learners collective standpoint and apply two different

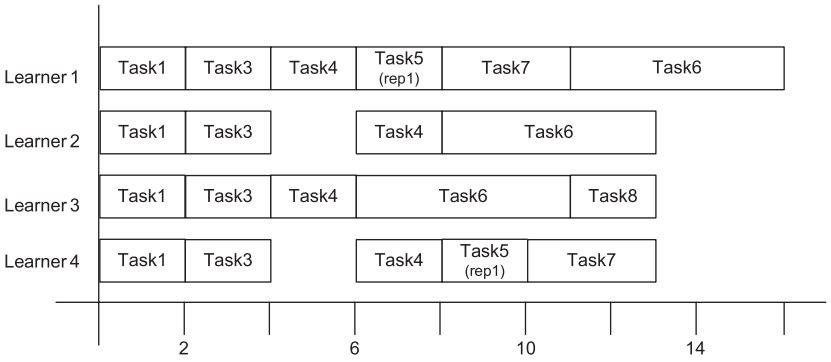

Fig. 6. Learning routes (plan) for each of the four learners when minimising the number of actions to be executed. Note that: (i) Task1 is executed at the same time by the four learners because of the synchronisation requirement of an in-person lesson and (ii) Task4 can be executed in parallel at most by two learners because of the capacity constraint.

optimisation criteria. First, we optimise the number of actions in the four learning routes, which means to find the solution with the min number of tasks (in a collective/joint way) to reach the learning goals, i.e. we formulate the metric as: minimise($\sum$ InPlan(task$_i$, learner$_j$)), $\forall$task$_i \in$ {Task1 ... Task9}, $\forall$learner$_j \in$ {Learner1 ... Learner4}. Second, we optimise the global makespan of the four learning routes which means the routes that lead to the shortest collective makespan, thus formulating the metric as: minimise(End), where End represents the last action of the plan.

5.2. Implementation and results

The constraint programming formulation is modelled and solved by Choco.[5] Choco is a Java library for constraint satisfaction problems, which, by means of the implementation of a class problem, allows the user to create variables and post *n*-ary constraints in a constraint network. Choco performs a backtracking search by selecting a value for each variable of the model and propagates such a value throughout the constraint network, updating the lower and upper bounds of other variables.

In our implementation, 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. During the solving process Choco tries to find the best solution according to the optimisation metric, i.e. given a certain deadline Choco performs an exhaustive search of solutions until no feasible solution improves the quality of the best solution found within the deadline or the allocated time is exceeded. 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. In general, Choco finds a solution very quickly and, in some cases, this turns to be the optimal one, though this cannot be generalised.

Our experiments have been conducted on a 2.60 GHz. Pentium IV with 1280 Mb. of RAM running Fedora Core 4 Linux and censored after 20 min. Particularly, if we focus on minimising the number of actions, Choco finds two solutions, the first one with 21 tasks and the second one with 20, which is depicted in Fig. 6. The first solution takes less than 2 s, the second one takes nearly 9 min and, after 20 min no better solution is found. On the other hand, if we focus on minimising the makespan Choco finds three solutions of length 17, 16 and 15 (see the resulting plan in Fig. 7) in 1.7, 2.2 and 2.8 s, respectively, exhausting the search space and thus guaranteeing that no better solution exists in less than 4 s.

[5] Choco can be downloaded from http://choco.sourceforge.net.

Actually, as can be seen in both figures the two plans are nearly identical and the only difference appears for Learner1, who in Fig. 6 takes one more action despite trying to minimise the number of actions in the plan. This indicates that the solver was unable to find the optimal solution within the allocated time. Choco had to explore, practically, the whole search space and this is because of the metric used, minimising the number of actions. As the cost of each action is exactly the same (one cost unit per action), the pruning mechanism has very little information to discriminate between alternatives and the search worked mostly in a breadth first fashion, which is very expensive. On the contrary, when minimising the makespan, Choco uses a more precise upper bound, thus pruning many partial solutions and exhausting the search space in a few seconds. In this case, since the cost of each action is different (action duration), the makespan of the solutions also differs between them and, consequently, the pruning mechanism is much more informed, making it possible to discard more parts of the search space. Thereby, the metric to be optimised can affect the complexity of the solving process, making it necessary to explore a bigger or smaller search space. Additionally, the use of heuristics can also play a valuable role during the search. The main advantage of this approach is that one can include different heuristics in the CSP solver with no changes in the constraint formulation at all.

6. Conclusions

Constraint programming formulation is a very appropriate approach to tackle planning problems that require the representation and management of a wide range of complex constraints, as it is the case of e-learning environments. Designing a learning route can be seen as generating a plan in a domain where it is necessary to attain learning outcomes (concepts) while handling resources and their capacity, synchronised tasks among learners of different profiles, 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 centre viewpoint, so it is important to guarantee good-quality, and even optimal, 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

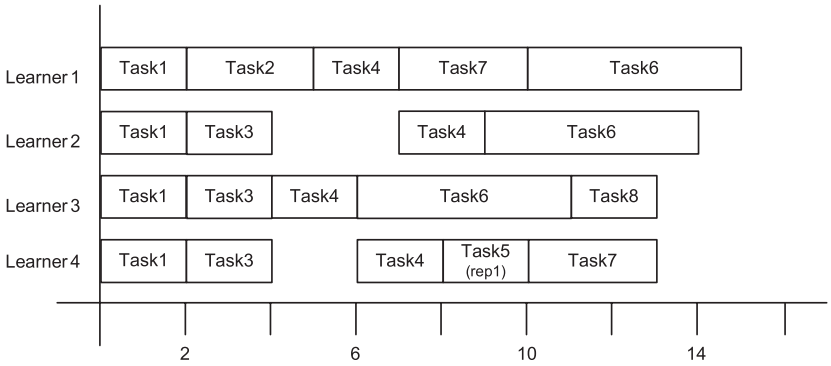

Fig. 7. Learning routes (plan) for each of the four learners when minimising the makespan of the plan. Note, as in Fig. 6, the constraints on Task1 and Task4.

and outcomes of each task by means of concepts, identify the profiles for which the task is best focus, determine the assessment points, establish the competence level for the concepts, etc. In order to make this task easier we provide a visual authoring tool that allows the instructor to simply drag and drop elements from a tool bar for elaborating such a design in a simple and intuitive way. As an additional advantage, since the course design comprises the specification of a general, flexible learning domain, it can be used for the generation of learning routes in many different contexts (teaching centres) with different time and resource constraints, for different learner profiles and with different optimisation criteria. In other words, it is worth the effort devoted to course design in favour 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 modelling 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. As a consequence of this, we are currently working on two aspects. First, we are studying other heuristic functions to make the solving process more efficient in order to face bigger problems and improve the scalability of the approach. Second, we are extending the authoring tool, as presented in Onaindía et al. (2007), to make the definition of the course design even more intuitive by splitting it into three steps: (i) define a conceptual design with the relations among concepts, (ii) define an instructional design with the tasks that achieve the concepts, and (iii) define a planning design with all the elements presented in this paper such as profiles, concepts, tasks, competence levels as percentages and complex constraints.

Acknowledgements

This work has been partially supported by the Spanish government projects MEC TIN2005-08945-C06-06 (FEDER) and Consolider-Ingenio 2010 CSD2007-00022, and by the Valencian government project GV06/096.

References

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

Bacchus, F., van Beek, P., 1998. On the conversion between non-binary and binary constraint satisfaction problems. In: Proceedings of the 15th National Conference on Artificial Intelligence. Madison, Wisconsin.

Bonet, B., Geffner, H., 2001. Planning as heuristic search. Artificial Intelligence 129, 5–33.

Do, M.B., Kambhampati, S., 2001. Sapa: a domain-independent heuristic metric temporal planner. In: Cesta, A., Borrajo, D. (Eds.), Proceedings of the European Conference on Planning (ECP-2001), pp. 109–120.

Felder, R.M., Silverman, L.K., 1988. Learning and teaching styles in engineering education. Engineering Education 78 (7), 674–681.

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

Gerevini, A., Long, D., 2006. Plan constraints and preferences in PDDL3. In: Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS-2006)—International Planning Competition, pp. 7–13.

Haslum, P., 2006. Improving heuristics through relaxed search—an analysis of TP4 and HSP*a in the 2004 planning competition. Journal of Artificial Intelligence Research 25, 233–267.

Hoffmann, J., Nebel, B., 2001. The FF planning system: fast plan generation through heuristic search. Journal of Artificial Intelligence Research 14, 253–302.

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.

Long, D., Fox, M., 2001. Encoding temporal planning domains and validating temporal plans. In: Proceedings of the 20th UK Planning and Scheduling SIG Workshop, pp. 167–180.

Manouselis, M., Sampson, D., 2002. Dynamic knowledge route selection for personalised learning environments using multiple criteria. In: Proceedings of the Intelligence and Technology in Educational Applications Workshop (ITEA-2002), Innsbruck, Austria.

Mittal, S., Falkenhainer, B., 1990. Dynamic constraint satisfaction problems. In: Proceedings of the National Conference on Artificial Intelligence, pp. 25–32.

Onaindía, E., Sapena, O., Garrido, A., 2007. An AI planning-based approach for automated design of learning routes. In: Proceedings of the 6th European Conference on e-Learning (ECEL-2007), pp. 453–461.

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

Penberthy, J., Weld, D.S., 1992. UCPOP: a sound, complete, partial-order planner for ADL. In: Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning. Kaufmann, Los Altos, CA, pp. 103–114.

Smith, P.L., Ragan, T.J., 2005. Instructional Design, third ed. Wiley, New York.

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

van Marcke, K., 1992. Instructional expertise. In: Proceedings of the 2nd International Conference on Intelligent Tutoring Systems, vol. 608. Springer, Berlin, pp. 234–243.

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

Vidal, V., 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., Anagnostopoulos, D., 2007. PASER: a curricula synthesis system based on automated problem solving. International 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.

Weld, D., 1994. An introduction to least commitment planning. AI Magazine 15 (4), 93–123.

Zimmerman, T., Kambhampati, S., 1999. Exploiting symmetry in the planning-graph via explanation-guided search. In: Proceedings of the National Conference on Artificial Intelligence.