Solving Clustered Oversubscription Problems for Planning e-Courses *

Susana Fernández and **Daniel Borrajo**
Departamento de Informática
Universidad Carlos III de Madrid
Avda. de la Universidad, 30. Leganés (Madrid). Spain
sfarregu@inf.uc3m.es, dborrajo@ia.uc3m.es

Abstract

In a general setting, oversubscription in planning can be posed as: given a set of goals, each one with a utility, obtain a plan that achieves some (or all) the goals, maximizing the utility, as well as minimizing the cost of achieving those goals. In this paper, we present an application domain, automatic generation of e-learning courses design, that shows a variation of the oversubscription problem. Here, there is only one goal: generating a course design for a given student. However, in order to achieve the goal, the course design can include or not different kinds of activities, each one with a utility and cost. Furthermore, these activities are grouped into clusters, so that at least one of the activities in each cluster is needed, though many more can be used. Finally, these problems also have an overall cost threshold (usually in terms of student time). So we show several techniques for solving the clustered-oversubscription problem and their impact on planning.

Introduction

The evolution of automated planning can be seen from three different interrelated perspectives: planning techniques, expressiveness of planning languages, and applications. In relation to the first one, it is one of the most active areas, since very powerful domain-independent techniques have been devised in the last two decades. However, we can see that there is no such counterpart in the other two. The planning competition has been one of the major drivers of this advancement, and the standard language, PDDL, one of its bases. But, on the one hand, PDDL started far from the expressiveness of previous planning languages used by specific planners, as O-Plan (Currie & Tate 1985) (in the HTN framework), or Prodigy (Veloso *et al.* 1995) (in the state space search framework). Though it has incorporated along the way some needed features that were already present on those planners (as handling metrics or definition of axioms), it continues to be still a bit behind in some aspects (as the explicit definition of resources, hierarchical knowledge, control knowledge definition, or the use of arbitrary functions in the preconditions and effects) and ahead on some others (as handling durative and continuous actions, or preferences). On the other hand, even if PDDL is gaining in expressiveness, very few planners can handle the full set of features that it defines. This is due mainly to the competition: if we do not focus on specific aspects of the planning task, then it is hard to see which technique is better for solving each aspect. This can be seen, for instance, on the latest focus of the optimal track on fixed static cost values in the actions for the metric to be optimized. Unfortunately, most real world applications require the use of some features of PDDL that are not handled by most planners, as well as the use of some representation features that are not present in PDDL. So, in relation to the third perspective, when trying to solve different real world applications, then one is very restricted to the planners that can be used. An example is the need to reason with state-dependent fluents that affect the computation of the metric (they belong to the formula that computes the metric). We could define state-dependent fluents as those whose value could be increased or decreased by the value of another fluent, whose value can increase or decrease at planning time. This could be partially handled by some planners (even "old" ones), but it cannot be handled by most current planners, and it is hard to be transformed into static-fluents.

Another relation between current planners and their ability to directly be used in applications is that most planners are still in an advanced prototype status. So, as we will see later, they either crash or fail on solving problems in domains that do not belong to the competition. This is probably due to small implementation errors, but this make it unfeasible to use them "as is" in current applications. Thus, there is still a gap between state of the art planners and their direct application. This can be easily explained given the two different emphasis on both communities: either generating powerful search techniques that can handle some expressiveness, and the need of polished implementations to be used as "off-the-shelf" components for applications.

Despite these problems, the application of planning techniques to real problems, sometimes, requires solving interesting associated problems that can be useful in more general contexts. This is the case of the work presented in this paper. The application area is the generation of learning designs adapted to students' profiles. And, the associated problem is a variation of the oversubscription problem

*This work has been partially supported by the Spanish MICINN under projects TIN2008-06701-C03-03 and TIN2005-08945-C06-05.

(OSP) that we have called clustered oversubscription. OSP was first introduced by (Smith 2004) and the objective of the planning process is not to find a feasible plan accomplishing all the goals, but to find a plan which reaches some of them. Usually, a plan quality metric, based on the different utilities of goals, is used in order to compare different plans achieving different sets of goals. The contribution of the current paper is twofold: modelling e-learning tasks as clustered-oversubscription planning problems; and providing two generic approaches for solving the resulting problems, applicable in other contexts than e-learning applications, as well. The remainder of the paper describes the e-learning application and how we have modelled it in PDDL. Next sections describe the two approaches that we have devised for solving the clustered-oversubscription problem by performing an action selection pre-processing to help the planning task: using linear programming and a search algorithm, the experiments performed to test the validity of the approach, the related work and the conclusions.

E-learning Planning Application

An e-learning task can be posed as: given a specific student, and a course defined by a pool of diverse learning activities, automatically generate a learning design for that student using the given learning activities. This domain has some interesting characteristics: (1) each learning activity belongs to a given cluster, and the final design should have at least one activity of each cluster, potentially more; (2) each activity has a utility, and a time (cost) that it takes to complete it; (3) the utility that a given learning activity will provide a student depends on some student features, so the use of conditional effects on actions is required; (4) as a hard constraint, the time to execute all learning activities in the final plan must be bound by a threshold (the total amount a student can dedicate to that course); (5) uncertainty should be handled in the learning design, given that the fact of whether the student will pass an exam or not in the middle of the course can affect the rest of the course; (6) activities can be easily seen as a hierarchy of tasks to be accomplished; (7) and, as with usual planning tasks, the activities present causal relationships among them (the student should follow some learning activities before others). Of all these features, we will focus in this paper on clustered oversubscription, and we will leave out (5) and (6). In clustered oversubscription, we perform some analysis before planning that selects a subset of the learning activities that is worth pursuing, given that they maximize utility (or have a high utility), and fulfill the constraint that the sum of their cost (time) is less than the cost limit (time threshold). We have used two different approaches for performing this selection: by posing the problem as a clustered-knapsack problem and then using linear programming; and heuristic search problem. Then, we translate this subset into two different planning knowledge as preferences or as the metric. Several planners are used to generate the final plan.

Our work is based on previous pedagogical research where they have tested a valid way for measuring the utility each learning activity reports to the students (Baldiris *et al.* 2008). So, we focus on the problem of sequencing learning activities in a course adapted to every student's profile by trying to maximize the total utility. Other relevant work concerning the use of AI planning and scheduling techniques for sequencing learning activities, according to different student's profiles and pedagogical theories, is reported in (Castillo *et al.* 2009; Brusilovsky & Vassileva 2003). But, they did not focus on the goal of finding a feasible plan that maximizes some measure of utility by reaching a subset of the goals. Our aim is that the proposed solution would be general enough so it can be extrapolated to other domains. An example of domain that have similar clustered-oversubscription problems is the rovers domain, not exactly the same as the one in the competition. The rovers need to take several samples, each sample of a specific kind (cluster) of rock, and there are several rocks of each kind. Also, taking those samples takes some time, and they have a specific time threshold to perform the activities. Other examples are earth observing satellites (clusters are areas with the same information, as big forests, mountains, or oceans) or the planning and optimization of resources in bus transportation networks (clusters are the different lines that compose the transportation network and the goal is optimizing their resources and minimizing costs).

Modelling the Planning Task in PDDL

The first step for building an e-learning planning application consists on modelling the courses. There are many theories, methodologies and tools for doing it (Jochems, Koper, & Merrienboer 2003; Cristea & Garzotto 2004). The most extended approach uses e-learning standards, as IMS-MD, where the designer of the course defines each different learning activity as an XML schema[1], named *learning object*.[2] For example, in the definition of an AI course, there may be a generic task for reading the introduction to planning. And, there could be several *learning objects* to accomplish it, as viewing a slide presentation, reading the introduction text from the text book, searching for the concept of Automated Planning in the Web and reading a couple of pages, or seeing a graph about planning. All the activities that achieve the task can be seen as belonging to the task cluster. It is enough for the student to follow any one of them to accomplish the task, but more than one can be performed as well, improving the overall utility (learning the subject). Each activity can be more or less appropriate to each student depending on the student profile (activity utility). So, the second modelling step is to determine this profile, given that it will define part of the knowledge of the planning problem. Again, there are many theories about it as the Felder's learning styles (Felder 1996).[3] From the planning point of view, the relevant issue

[1]There are tools as Reload, http://www.reload.ac.uk/, that help on creating these schemas.

[2]The full standard can be found in the IMS Global Learning Consortium web page: http://www.imsglobal.org

[3]The learning styles model developed by Richard Felder incorporates four dimensions: the Perception dimension (sensitive/intuitive), the Processing dimension (active/reflective), the Input dimensions (visual/verbal) and the Understanding dimension (sequential/global).

is that each learning activity can be labelled with an utility value that depends on the particular characteristics of each student. Each activity also takes a time to fulfill it. And, also, learning activities can have dependency relations among them. For instance, before reading about PDDL, the student should have some knowledge on predicate logic.

Once someone has defined a course in some e-learning standard, we use a compiler that translates it into PDDL. The input to the planner in the e-learning application is a set of learning activities defined in the IMS-MD standard (domain) and a student profile according to the Felder's learning styles (problem). The IMS-MD set of activities are translated into a PDDL domain and the student profile is represented as predicates included in the PDDL problem. One domain is defined per course. We use one predicate for each Felder's learning style. For example, *(sequential ?s - student ?p - profile_level_type)*. The *profile_level_type* can take the value *strong, moderate* or *balanced*. If the system determines that the student is, for example, *strong sensitive* and *strong active* we would add in the *:inits* of the PDDL problem the predicates *(active student1 strong)* and *(sensitive student1 strong)*.

Each *learning object* represents a learning activity and is translated into a PDDL action in the following way:

- The XML label <title> is used as the action name.
- We define a predicate with the same action name, but adjoining the prefix *task_* and the suffix *_done*. It is added to the action effects and represents the fact that the student has performed such activity and prevents him/her from repeating it.
- The XML label <typicallearningtime> represents the activity duration. We use a fluent, *(total_time_student ?s)*, that is increased in the amount of this label in the action effects.
- The XML label <learningsourcetype>. According to pedagogical theories, each learning source type is related to the Felder's learning styles (Baldiris *et al.* 2008). We have used a fluent, *(reward_student ?s)* (action's utility), and conditional effects to represent this relation. For example, when the learning source is a *lecture* this pedagogical theory says that is *very good* for *reflective, intuitive* and *verbal* students. So we add the following conditional effects to the PDDL action:

```
(when (reflective ?s strong)
      (increase (reward_student ?s) 40))
(when (intuitive ?s strong)
      (increase (reward_student ?s) 40))
(when (verbal ?s strong)
      (increase (reward_student ?s) 40))
```

  To compute the increasing values of the *reward_student*, we use a table defined in (Baldiris *et al.* 2008) where rows represent learning source types, columns Felder's learning styles and intersections can take the values: *very good*, *good* or *indifferent*. It has fifteen rows (*lecture, narrativetext,. . . ,exam*) and eight columns (*Active, Reflective, Intuitive, Sensitive, Sequential, Global, Visual* and *Verbal*). We assume each activity can have a maximum reward of 120, and that a *very good* value takes double reward than a *good* value and an *indifferent* value has zero reward. For each learning source type in a row, we sum the number of columns with a value different from *indifferent*, d. The reward increases for each learning style in that row will be $120/d$ for *very good* and half of it for *good*. For example, the *index* row in the table takes the *very good* value for the *Global* column, *good* for *Sequential* and *Verbal* and *indifferent* for the rest of the columns (i.e. there are three columns with a value different from *indifferent*). Thus, the reward when the student is strong *global* will be 120/3=40 and half of it when the student is strong *sequential* or *verbal*. We assume all activities provide some positive reward to the student, so a minimum of five is added to the total reward.
- The XML label <relation>. We will use two of the several types of relations defined in IMS-MD: *Requires* and *IsBasedOn*. [4] *Requires* represents a causal dependency between two activities, so each activity in the relation is translated as an action precondition. *IsBasedOn* represents that the activity can be performed by doing one of the activities in the relation and they are translated as *or* preconditions and fictitious activities (given that they will not be used on the oversubscription computation). In fact, a learning object with an *IsBasedOn* relation is considered as a fictitious action, because the student has to perform only one of the actions in the *or* condition and both the reward and the total_time remain the same.

Figures 1 and 2 show PDDL actions translated from learning objects with relations of type *Requires* and *IsBasedOn* respectively. The first action describes the activity *simulates-strips-problem*. It requires that the student has already performed activity *reads-classical-planning*, it takes 30 minutes, and it adds the corresponding rewards. The learning source type is *problem* that is *very good* for strong *active, sensitive* and *visual* students and *good* for strong *global* students. We add precondition `(not (task_strips_done ?s))` to avoid including twice the same activity in the plan. The second action represents that a student could perform the activity *simulates-strips-problem* or *experiments-strips-problem* to accomplish the task *task_strips_done*.

The last learning object defined in the IMS-MD design is named `fictitious-finish-course` and it contains, as a *Requires* relation, the tasks required to fulfill the course. This learning object is translated into a fictitious PDDL action with one effect, *(task_course_done ?s)*, and its preconditions are the tasks required to complete the course, plus the predicates *(< (total_time_student ?s) (time_threshold_student ?s))*, to avoid the plan to exceed the time limit, and *(> (total_reward_student ?s) (reward_threshold_student ?s))*. Both thresholds are defined in the problem file, *(time_threshold_student ?s)* is the total time the student can devote to the course and *(reward_threshold_student ?s)* is computed from the activities selection explained in the next section. The problem file has

[4] There are two more kinds of relations: *IsPartOf* that could be used by HTN planners, and *References* for recommending previous learning objects.

```
(:action simulates-strips-problem
   :parameters (?s - student)
   :precondition (and (task_reads-classical-planning_done ?s)
                      (not (task_simulates-strips-problem_done ?s)))
   :effect (and (task_simulates-strips-problem_done ?s)
                (increase (reward_student ?s) 5)
                (increase (total_time_student ?s) 30)
                (when (active ?s strong)
                      (increase (reward_student ?s) 30))
                (when (sensitive ?s strong)
                      (increase (reward_student ?s) 30))
                (when (global ?s strong)
                      (increase (reward_student ?s) 15))
                (when (visual ?s strong)
                      (increase (reward_student ?s) 30))))
```

Figure 1: Example of PDDL action translated from a learning object with a *Requires* relation.

```
(:action OR-fictitious-strips
   :parameters (?s - student)
   :precondition (and (not (task_strips_done ?s))
                      (or (task_simulates-strips-problem_done ?s)
                          (task_experiments-strips-problem_done ?s)))
   :effect (and (task_strips_done ?s)))
```

Figure 2: Example of PDDL action translated from a learning object with a *IsBasedOn* relation.

only the goal *(task_course_done ?s)*. This representation allows that any planner that supports full ADL extension (including conditional effects) and fluents could find a solution, but not the *best* solution, i.e. the sequence of actions that reports the maximum reward to the student. We could use as plan metric to minimize the total_time, but then the planners will not maximize the reward. On the other side, a metric for maximizing the reward is unfeasible for planners based on the enhanced relaxed-plan heuristic introduced in Metric-FF (Hoffmann 2003), as they only work with minimization tasks. We cannot easily transform the metric for maximizing the reward into a metric for minimizing the inverse using state-independent fluents, due to how rewards are computed in this domain. We used the table defined in (Baldiris *et al.* 2008) that contains information on whether the learning resource type is *good*, *very good* or *indifferent* for each learning style. But we cannot assert anything about the inverse. For instance, it is not true that if a *lecture* is *very good* for a reflective student it is *very bad* for a non reflective student. It could be that for a non reflective student the *lecture* is *indifferent*. So, we do not know which values to assign to those cases not in the table. Others equivalent transformation would require state-dependent fluents for computing the metric. Thus, instead of giving the planners the task of minimizing a function of time and inverse of reward, we first run an activity (action) selection algorithm to select the most promising activities, $O = \{a_i\}$, and then use that information when planning.

Activities Selection

This section describes two methods for obtaining the set of actions that maximizes the student reward within a time limit, by using linear programing or heuristic search. First, we formalize the problem and then we explain each approach.

Problem Formalization

The problem is similar to the well-known knapsack problem[5] in combinatorial optimization, but with the addition of clusters. We have a set of activities, A, each with a time (duration, cost) and a reward (utility), $\forall a \in A, a =< t, r >$. The goal is to determine the set of activities to include in the output, $O = \{a_1, \ldots, a_n\}, a_i \in A$, so that the total time is less than a given limit, $T, \sum_{a_i=<t_i,r_i>\in O} t_i \leq T$, and the total reward is maximized. Besides, activities are grouped into a set of clusters, $C = \{c_1, \ldots, c_m\}, c_i = \{a_1, \ldots, a_{c_i}\}$ that can perform the same learning task. The solution must contain at least one activity of each cluster, $\forall c_i \in C$ at least one $a_j \in c_i$ should be in O. We automatically extract this maximization task from a PDDL domain and problem file. Domain actions translated from learning objects with a *IsBasedOn* relation define the clusters. The activity reward is computed from the problem file (to extract the student learning-styles) and the conditional effects concerning the fluent *(reward_student)*. And, the activity time is the increase of the fluent *(total_time_student)* in the corresponding PDDL action.

Linear Programming

The linear programming (LP) task is defined by: the set of activities A; the set of clusters or tasks C; the time t_i and reward r_i of each activity; a binary matrix $c_{i,j}$ representing whether activity i belongs to task j; the time limit T; and one binary variable x_i for each activity representing whether activity i is in the solution set, O. The objective function consists of maximizing $\sum_i x_i \times r_i$ subject to two linear inequality constraints: $\sum_i x_i \times t_i <= T$ for the time limit and $\sum_i c_{i,j} \times x_i >= 1$ to ensure that at least one activity is present per cluster. This last constraint is the new one with respect to standard knapsack problems. An important property is that LP algorithms guarantee optimality. Another alternative would have been modelling the action preconditions as linear equality and inequality constraints. However, this is still an open issue and it is not trivial due to the disjuntion relations among some preconditions. Therefore, a complete formulation of the whole problem in LP or CSP is difficult. Here, we opt for a hybrid approach in that LP solves the LP component of the task, and planning solves the causal component of the task.

Heuristic Search

The search problem can be modelled as:

- States: $S = (\{AC\}, t, r)$ where $\forall ac \in AC, ac = \{(a_i, c_i)\}$ are the activities in the solution set, together with the clusters they belong to, t is the sum of the time of the selected activities and r the sum of their rewards.
- Initial state: $S = (\emptyset, 0, 0)$.
- Goal function: whether there is at least one activity per cluster and $time \leq T$, where T is the time limit.

[5]Given a set of items, each with a weight and a value, determine the items to include in a collection, so that the total weight is less than a given limit and the total value is as large as possible.

- Actions: there is only one action that adds a new activity $a_i =< t_i, r_i >$ and its cluster to state $S = (AC, t, r)$, given that $t_i + t \leq T$ (it does not pass the time limit).

We use a hill-climbing algorithm with backtracking, minimizing the evaluation function, similar to the one used in A*: $f(s) = g(s) + h(s)$. $g(s)$ is the cost of the last activity added to s computed with equation 1:

$$g(s) = (2 - Nr(r)) \times (0.1 + Nt(t)) \quad (1)$$

where t is the time of the activity, r its reward, $Nr(x)$ and $Nt(x)$ are functions to normalize the reward and the time respectively. We do $(2 - Nr(r))$, since the algorithm tries to minimize $f(s)$, so that we avoid $g(s)$ takes value 0. We do $(0.1 + Nt(t))$ also to avoid $g(s)$ takes value 0. $h(s)$ is an estimation of the remainder cost to reach the solution. To compute $h(s)$, we compute the clusters C' that are not yet included in s, and the minimum cost of the activities belonging to each of those clusters, $\forall c_j \in C', min - c_j = \min_{a_i=<t_i,r_i>\in c_j} g(t_i, r_i)$. $h(s)$ is the sum of these $min - c_j$. We used hill-climbing with backtracking that is complete but not optimal for comparison (defining it as an appropriate task for an admissible technique as A* is harder, given that we are handling two metrics, and we have to establish some end criterion based on only one of them).

Using the Selection when Planning

We have devised two approaches for using the knowledge on the most promising activities when planning, by including the actions in O as PDDL3 preference-goals or in the plan metric. In the former, we add the goal *(preference p_i (task_a_i_done student1))* for each $a_i \in O$ (p_i is the preference identifier). In the latter, we define a new predicate *(action-in-plan ?s - student ?a - plan-action)* and a new fluent *(penalty ?s -student)*. Each domain action a_i is updated with the following conditional effect (being <action-name> its name):

```
(when (not (action-in-plan ?s <action-name>))
      (increase (penalty ?s) 1))
```

Thus, everytime an activity is used in the plan that does not belong to the selected set, the penalty is increased. Then, we add a predicate *(action-in-plan student1 a_i-name)* in the problem file for each $a_i \in O$, and we use the metric *(minimize (penalty student1))*. Thus, the planner will only try to minimize the number of activities in the plan that were not selected previously. In short, O contains the knowledge for maximizing the student reward, but it lacks the knowledge for sequencing the activities, and this is the role of the planner.

The action selection methods also compute the maximum reward, $MaxR$, that could be obtained for the given time limit in case there were no activities dependencies:

$$MaxR = \sum_{a_i=<t_i,r_i>\in O} r_i \quad (2)$$

On the other hand, the *penalty* metric represents the number of activities that appear in the plan, but were not selected by the selection mechanism (due to the causal dependencies). Therefore, we can compute a lower bound value for the reward threshold necessary for the precondition *(> (total_reward_student ?s) (reward_threshold_student ?s))* of the last domain action with equation 3.

$$reward\_threshold\_student = MaxR - (120 \times penalty) \quad (3)$$

(120 is the highest activity reward).

Experiments

We have carried out the experiments with a domain representing a real Artificial Intelligence course. The domain has 125 actions and there are 52 tasks or clusters. Each task includes between only one activity up to six. There are activities for covering all the typical tasks in an AI course as reading a subject, practising, programming, performing exams, ... The first experiments aim to test the validity of linear programming and heuristic search for obtaining the set of activities that maximizes the student reward within a time limit. We want to compare two approaches, with and without optimality guarantee, to test whether the non optimal approach could produce acceptable plans as well or not. We have used the GLPK (GNU Linear Programming Kit) package[6] and our own search-algorithm library implemented in Lisp (also available in the Web). We have tested the linear programming and search algorithms with different activities subsets, starting with only the activities included in one cluster up to all the clusters. We have set the time limit as the sum over the considered clusters of the time of the highest-time activity in each cluster. We also used the sum of the rewards of those activities as a lower-bound reward for comparison. Figure 3 shows these results. LP finds optimal solutions while the distance between the search algorithm solutions and the LP ones increases with the number of clusters. The LP algorithm always found the solution in less that 0.1s while the search-algorithm execution time steady increased from 0.1 up to 8s.

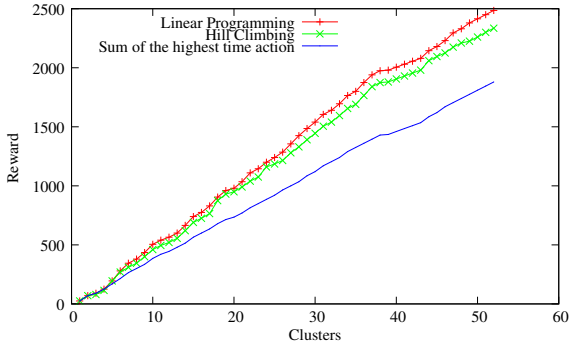

Figure 3: Reward when time limit is the sum of the time of the highest-time activity in each cluster.

In the next experiments we varied the time limit by decreasing it in 20%, 15%, 10% and 5% and also increasing

[6] http://www.gnu.org/software/glpk/

it in the same amounts. Figure 4 shows the reward results using all the clusters. We set time increment 0 (the comparison value) as the sum of the times of the highest-time activity in each cluster (it corresponds to 3810). We decrease and increase it from a -20% (3048) to a 20% (4572). As expected, the hill-climbing algorithm finds solutions worse than the optimal solution found by LP. With respect to execution time, LP is usually faster than the search algorithm, but there are some cases where LP takes a long time to converge. We repeated the experiments with different number of clusters obtaining similar results. Anyway, the execution time was never higher than 18 s.

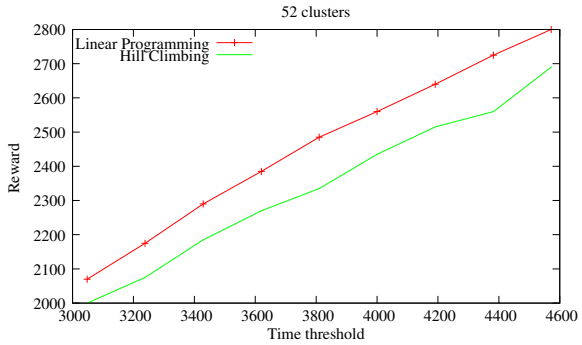

Figure 4: Reward of 52 clusters when time limit varies from -20% up to 20%.

We have used SGPlan6 (Hsu & W.Wah 2008) for incorporating the obtained knowledge by the LP and search algorithms as PDDL3 preferences-goals. And we have used the CBP (Fuentetaja, Borrajo, & López 2009) planner for incorporating the obtained knowledge as the plan metric. SGPlan6 is the only planner (from the IPC-08) that supports PDDL3 and it was able to solve our problems. The CBP planner is based on Metric-FF, but with some enhancements (specific cost-based heuristic function, lookahead search) that allow it to solve our problems using plan metrics. Metric-FF was able to solve the problems, but only with EHC. BFS, which was needed for using plan metrics, always exhausted computer memory. The other planners that competed in the IPC-08 could not solve our problems. We believe that this might be due to the domain size and the PDDL features included in the domain (fluents, numerical expressions in the preconditions, preconditions with *or* and *not* expressions and conditional effects).

Next, we compare the following configurations: i) EHC: original Enforced Hill-climbing algorithm in Metric-FF, ii) CBP-BFS TIME: CBP planner with Best First Search algorithm and minimizing the *(total_time_student)* as plan metric, iii) CBP-BFS LP: CBP planner with BFS algorithm incorporating the knowledge obtained by the LP algorithm and minimizing *(penalty_student)* as plan metric, iv) CBP-BFS HILL: the same as before but incorporating the knowledge obtained by the search algorithm, v) SGPLAN6: SGPlan algorithm without preference goals, vi) SGPLAN6 LP: SGPlan using as preference goals the activities obtained by the LP algorithm, vii): SGPLAN6 HILL: SGPlan using as preference goals the activities obtained by the search algorithm. Also, CBP is able to find multiple solutions. So, we let it run during one second in multiple solutions mode, and it usually found two solutions. We have experimentally seen that increasing this running time does not make it find more solutions, but it exhausts computer memory.

As mentioned in the previous section, the pre-process offers us two kinds of information: computing the most promising activities and the reward threshold. But, the reward-threshold computation needs the *penalty* values that CBP-BFS LP and CBP-BFS HILL can measure using the most promising activities. So, first, we executed CBP-BFS LP and CBP-BFS HILL with no *(reward_threshold_student)* value, and different time limits, column TL, (the ones used in the experiment showed in Figure 4) to measure the *penalty*. Table 1 shows the results for both systems. Column P is the metric value *penalty* (number of activities that appear in the plan, but were not selected by the selection mechanism). Column R is the cumulative reward of the actions in the plan. And, column T is the cumulative time of the actions in the plan. There is one row for each time limit and for each solution CBP-BFS found during one second in multiple solution mode. Apart from the *penalty*, the table shows that there are some cases where CBP-BFS HILL obtained less reward (for the same TL), although the metric had a better value (lower value), and the planner finds better solutions. On the contrary, CBP-BFS LP always found better rewards for better metric values.

TL	CBP-BFS LP			CBP-BFS HILL		
	P	R	T	P	R	T
3048 (-20%)	5	1490	2510	8	1500	2540
3048 (-20%)	2	1545	2465	2	1460	2345
3238 (-15%)	5	1490	2510	8	1500	2540
3238 (-15%)	2	1545	2465	1	1415	2305
3429 (-10%)	5	1490	2510	8	1500	2540
3429 (-10%)	2	1545	2465	2	1460	2345
3620 (-5%)	5	1490	2510	8	1500	2540
3620 (-5%)	2	1545	2465	2	1460	2345
3810 (0%)	4	1505	2540	7	1500	2540
3810 (0%)	1	1560	2495	0	1415	2305
4000 (+5%)	4	1520	2570	8	1480	2540
4000 (+5%)	1	1575	2525	2	1460	2345
4191 (+10%)	4	1520	2555	7	1480	2540
4191 (+10%)	1	1575	2525	1	1460	2345
4382 (+15%)	4	1520	2555	7	1505	2555
4382 (+15%)	1	1575	2525	1	1430	2335
4572 (+20%)	4	1520	2555	5	1505	2555
4572 (+20%)	1	1575	2525	1	1475	2375

Table 1: Reward and time using the *penalty* minimization as plan metric without *(reward_threshold_student)* value.

Second, we computed the *(reward_threshold_student)* values from the activities selection and the penalty values (the lowest value) with equations 2 and 3 and executed CBP-BFS LP and CBP-BFS HILL using their corresponding reward threshold. Figures 5 and 6 represent the reward and time of the activities in the solution plan obtained by each configuration with the different time thresholds. We

have included only the best (highest reward) CBP solution for each configuration. We exclude SGPLAN6 HILL and SGPLAN6 LP from the results, because they ignored the time-limit constraint generating unfeasible plans. These figures show that the planning systems without using the pre-process information (EHC, SGPLAN6 and CBP-BFS TIME) always found the same solution although the time limit varied. CBP-BFS TIME obtained the plan with the lowest time to fulfill it but with the lowest reward value, as well. EHC and SGPLAN6 found the same solution with a reward value notably inferior to the reward in CBP-BFS LP and CBP-BFS HILL plans.

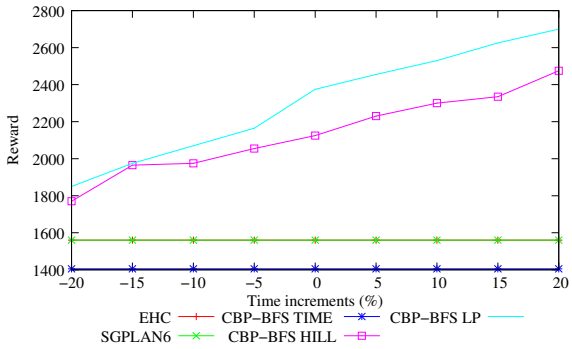

Figure 5: Total reward for different time limits using a reward threshold.

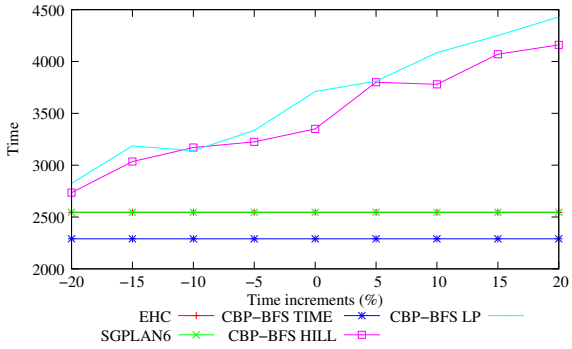

Figure 6: Total time for different time limits using a reward threshold.

The best plans are those with the maximum reward and the minimum time. We are interested in increasing the total reward the student achieves, but also in decreasing the total time s/he devotes to the course. To obtain a balance, we cannot directly subtract reward from time, because they have different scales. So, instead, we compute the product of the percentage of reward increment and the percentage of time decrement, both with respect to bounds. We set a lower bound for the reward as the sum of the rewards of the activities with highest time value in each cluster (1105 in this case), and the time limit is the upper bound for the time. Then, we compute the percentage of reward increment ($\triangle R$), with respect to the above lower bound and the percentage of time decrement ($\bigtriangledown T$), with respect to the above upper bound. Figure 7 represents the multiplication of the reward increment and the time decrement for the different time limits and configurations. Again, it shows that the pre-process notably increases the time/reward balance.

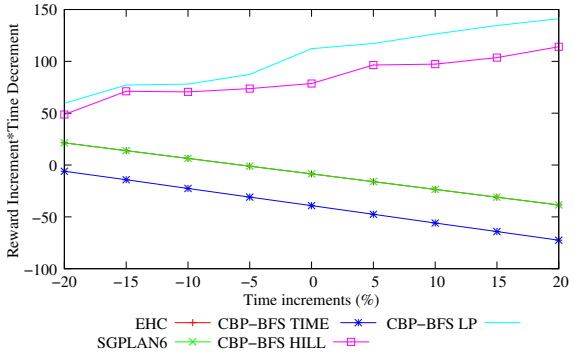

Figure 7: Reward increment multiplied by the time decrement for different time limits using a reward threshold.

Planning time is very small (less than half a second) for all the configurations. When EHC and CBP did not solve a problem within this short time they exhausted computer memory.

Finally, we wanted to test the approaches using only one of the two kinds of information the pre-process offers us. First, adding a *(reward_threshold_student)* value, assuming we were able to know an approximate value for it, but without computing the most promising activities. And second, computing the most promising activities but not the reward threshold. First, we executed EHC, CBP-BFS TIME and SGPLAN6 with the reward threshold information obtained in previous experiments. EHC could not solve any problem. SGPLAN6 had always the same reward (1560) and time (2545), worse than CBP-BFS LP and CBP-BFS HILL. And, CBP-BFS TIME obtained the same results as CBP-BFS LP, but there were two problems, with time limit increments of 0 and 5, that it could not solve. Second, we executed all the systems without the *(reward_threshold_student)* information and only CBP-BFS LP was slightly better than the original planners, METRIC-FF WITH EHC and SGPLAN. Therefore, only the combination of the two kinds of information that the pre-process offers us (computing the most promising activities and the reward threshold) reports a significant benefit to the solution plans. And, the LP pre-process is better than the hill-climbing one. CBP planner with BFS algorithm, minimizing the *(total_time_student)* as plan metric and using a reward threshold sometimes obtains good plans. But it has the drawback that it needs to compute that threshold. Besides, it cannot solve some cases (we do not have yet an explanation for this phenomenon).

Related Work

There are several classes of oversubscription problems, depending on the cause preventing all the goals to be accomplished. Our oversubscription problem is caused by having

limited resources (time), and plans achieving all goals would need more resources than the available ones. It is known as MAX-COST problems, focusing on maximizing plan utility, while maintaining the cost lower than a *maximum cost*, contrary to NET-BENEFIT problems, that focus on maximizing the *net benefit* which is the cumulative utility of the achieved goals minus the cumulative cost of actions used in the plan. In our case, time is the resource and performing all the activities defined in the course would be the problem goals. But not all goals can be achieved because of lack of time. The plan must select only those goals that report the maximum utility (reward) to the student without exceeding the given time threshold.

Goals selection procedures vary; in (Smith 2004) an orienteering problem (a generalization of the travelling salesperson problem (Feillet, Dejax, & Gendreau 2005)) is constructed and solved, obtaining an ordered subset of goals to plan for. A different approach is taken by (Sanchez Nigenda & Kambhampati 2005), which using relaxed plans heuristics (Hoffmann 2003) estimates the net benefit of including a new goal into the set of currently selected goals. We also select goals up-front, but instead of constructing an orienteering problem we construct a clustered-knapsack problem. In (Smith 2004) a beam search algorithm is used to solve the orienteering problem, while we use linear programming and a hill-climbing search algorithm to solve the clustered-knapsack problem.

Conclusions and Future Work

This paper presents our work on building an e-learning planning application for generating learning designs adapted to different students' profiles. An e-learning course is defined by a set of different learning activities, that use learning objects. Learning activities, with their duration, student profile's dependence and the relations defined in their metadata, keep a strong resemblance with actions as traditionally used in AI planning domains. In particular, each learning activity can be simply modelled as an action, its dependency relations as preconditions, and its outcomes as effects. This way P&S techniques can be very powerful to automatically generate sequences of learning activities fully adapted to the students. However, current planners are good for sequencing actions, but they are not so good for optimizing. Optimizing requires to use fluents and plan metrics, and it notably increases planning complexity. We are interested in finding the sequence that maximizes the student's net benefit. Therefore, we have modelled the problem as a clustered-oversubscription problem performing an action selection pre-processing to help the planning task. We have proposed two action-selection schemes, based on linear programing or heuristic search, that partially solves the optimization problem, and whose output is compiled into the planning task. It turns out that for this task, LP provides optimal solutions in a more than reasonable time. PDDL provides expressive power to model the problem, and to include the optimization knowledge in two different formats. The resulting PDDL domain is apparently simple, because there are no delete effects, but the complexity arises from the number of actions (125), the conditional effects, and the plan metric. Only the SGPlan6 and CBP planners were able to solve the problem.

In the future, we would like to test our approach in other domains as the *clustered* rovers and satellites ones, as well as, in other application areas. For example, the planning and optimization of resources in bus transportation networks.

References

Baldiris, S.; Santos, O.; Barrera, C.; J.G., J. B.; Velez, J.; and Fabregat, R. 2008. Integration of educational specifications and standards to support adaptive learning scenarios in adaptaplan. *Special Issue on New Trends on AI techniques for Educational Technologies. International Journal of Computer Science and Applications (IJCSA)*.

Brusilovsky, P., and Vassileva, J. 2003. Course Sequencing Techinques for Large-Scale Web-Based Education. *International Journal Continuing Engeenering Education and Lifelong Learning* 13(1/2):75–94.

Castillo, L.; Morales, L.; González-Ferrer, A.; Fdez-Olivares, J.; Borrajo, D.; and Onaindía, E. 2009. Automatic generation of temporal planning domains for e-learning problems. *Journal of Scheduling*. Accepted.

Cristea, A., and Garzotto, F. 2004. Adapt major design dimensions for educational adaptive hypermedia. In *ED-Media '04 conference on educational multimedia, hypermedia & telecommunications. Association for the advancement of computing in education*.

Currie, K., and Tate, A. 1985. O-plan: Control in the open planning architecture. In *BCS Expert systems conference*.

Feillet, D.; Dejax, P.; and Gendreau, M. 2005. Traveling salesman problems with profits. *Transportation Science* 39(2):188–205.

Felder, R. M. 1996. Matters of style. *ASEE Prism* 6(4):18–23.

Fuentetaja, R.; Borrajo, D.; and López, C. L. 2009. A look-ahead B&B search for cost-based planning. In *Proceedings of CAEPIA'09*.

Hoffmann, J. 2003. The Metric-FF planning system: Translating "ignoring delete lists" to numeric state variables. *JAIR* 20:291–341.

Hsu, C.-W., and W.Wah, B. 2008. The sgplan planning system in ipc-6. In *Working notes of the ICAPS'08 International Planning Competition*.

Jochems, W.; Koper, R.; and Merrienboer, J. V., eds. 2003. *Integrated E-Learning: Pedagogy, Technology and Organization*. Kogan Page, Limited.

Sanchez Nigenda, R., and Kambhampati, S. 2005. Planning graph heuristics for selecting objectives in over-subscription planning problems. In *Proceedings of ICAPS-05*, 192–201.

Smith, D. 2004. Choosing objectives in over-subscription planning. In *Proceedings of ICAPS-04*, 393–401.

Veloso, M.; Carbonell, J.; Pérez, A.; Borrajo, D.; Fink, E.; and Blythe, J. 1995. Integrating planning and learning: The PRODIGY architecture. *Journal of Experimental and Theoretical AI* 7:81–120.