View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by Bulgarian Digital Mathematics Library at IMI-BAS

MANAGING INTERVAL RESOURCES IN AUTOMATED PLANNING

V.Poggioni, A.Milani, M.Baioletti

Abstract: In this paper RDPPLan, a model for planning with quantitative resources specified as numerical intervals, is presented. Nearly all existing models of planning with resources require to specify exact values for updating resources modified by actions execution. In other words these models cannot deal with more realistic situations in which the resources quantities are not completely known but are bounded by intervals. The RDPPlan model allow to manage domains more tailored to real world, where preconditions and effects over quantitative resources can be specified by intervals of values, in addition mixed logical/quantitative and pure numerical goals can be posed. RDPPlan is based on non directional search over a planning graph, like DPPlan, from which it derives, it uses propagation rules which have been appropriately extended to the management of resource intervals. The propagation rules extended with resources must verify invariant properties over the planning graph which have been proven by the authors and guarantee the correctness of the approach. An implementation of the RDPPlan model is described with search strategies specifically developed for interval resources.

Keywords: AI, Automated Planning, Planning with resources, Propagation rule, Search strategies.

Introduction

Various models have been proposed for extending the pure logical classical planning models in order to manage more real world features. A very promising issue toward this goal is the research line which aims to provide the planners with the ability of planning with resources. In this framework, in addition to the logical relationships among domain objects, operators and states, the planning models are able to cope with quantitative aspects of the world, such as actions which involves consumable/reusable resources, domain constraints on resources, goals involving quantities.

Several planning models for resources management have been proposed for extending virtually all the most successful planner approaches; among them it is worth noticing models UCPOP—like models [4], Graphplan-like [12], SAT—like [15,16], and also HTN based approaches [5].

The types and features of the modelled resources are also varying from unary and discrete resources [13] to reusable resources [4] and conjunctive constraints over resources [6,9]. A different approach is that of planners specialised in the management of time, as a quantitative resource; these planners allow the management of extension such as propositions which holds over time intervals , actions with durations and complex time numerical constraints [6,10]. The issue of a complex time management is beyond the scope of this work.

It is worth noticing that the introduction of quantitative resources in a planning framework has brought into planning some typical issues of scheduling and CSP, such as optimisation search and constraints management. Moreover having quantitative resources also change the typical view a planning problem can be regarded to. At the simplest level there are “pure logical” problem goals which can be specified as in the classical framework , nevertheless the plan generation phase will have to take into account of resources precondition/effects; the problem can otherwise specify “mixed” logical/quantitative goals, or even “pure quantitative goals”, (e.g. consider the problem of finding a plan for the purely quantitative goals: Consume at least 100 calories, Produce 1 Billion profit etc.); finally pure/mixed “optimisation” goals can also be specified, where no logical or quantitative goals exist (e.g. consider the problem: “producing as much profit as you can”), this latter optimisation aspect has been incorporated in PDDL 2.1 [7] where it is possible to specify an object function to be optimised.

Models of planning with resources certainly represent an important step toward a more accurate model of the real world, but, on the other hand, most of the proposed models fail to give any account of the potential uncertainty which can affect the quantities related to resources. Many facts in a real world state can be described in a satisfactory way by a boolean proposition (e.g. (on A table) (open door) etc.), but it is not very realistic to assume that an exact number can model the continuous quantities describing a given resource. Most models of planning with resources allow to describe non exact quantities in preconditions, such as, for example, an interval of values that the resources can assume in order make the action to executable (e.g. the fuel must be between 10 and 30, the voltage must be between 210 and 230, this preconditions can be

modeled both in [12] as well in [9]), surprisingly the same planning models do not allow to specify intervals of values in action effects. In fact models of planning with resources admit updates and assignment operations which allow functional quantities (for example consumed_fuel can be functionally computed by distance * fuel_consumption) but where the increment of the current resource level is a single well determined numerical value (e.g. *consumed_fuel = distance * fuel_consumption = 12 * 0.25 =3* that is a single value) [7,9,12]. Indeed, it seems to be an apparent contradiction that the semantics of preconditions can be also given in terms of non exact quantities, while the semantics of effects have to be given only in term of precise values.

In this work we show how this gap between non exact preconditions and fully specified effects can be bridged by specifying in both cases quantities varying over intervals. RDPPlan, the model of planning with resources which we will describe, is based on DPPlan [1], a planner which uses a non directional search algorithm on the planning graph. RDPPlan is compatible with the resources model as described in standard PDDL 2.1, and extends it by allowing updates and assignments of quantities specified by intervals.

In the following paragraphs, after recalling the main features of DPPlan, it is introduced RDPPlan, the planning model with resources, showing that the addition of resources management does not have a great impact on the overall DPPlan approach and on the planning graph structure. Resources management in RDPPlan is realized by the modification of propagation rules and the introduction of appropriate rules for failure detection caused by resources constraints violation. Moreover, the structure and the algorithms we provide for resources management can be easily extended for update operations which operate over intervals, as shown in the fourth paragraph.

It is worth noticing that RDPPlan architecture is not committed to any particular search strategy (i.e. the resources management and failure detection is embedded into the propagation rules), the consequence is that search strategies can be easily added to RDPPlan. Strategies specifically developed for resources are described in paragraph five.

Examples and experimental results show that this approach seems to be appropriate for modeling real world situations where the consumption/production of resources cannot be expressed by a single value (for example, assuming that f*uel_consumption per Km* is a non exact quantity which ranges between *[0.25, 0.33]*, then *consumed_fuel* can be calculated as an interval by computing distance * *fuel_consumption = 12 * [0.25,0.33] = [3, 4]*).

Finally we point out some possible topics which are worth to be investigated in the RDPPlan framework, such as further strategies and heuristics for problems with resources, and further extensions to the resource model, like managements of fuzzy quantities.

DPPlan and its propagation rules

DPPlan is mainly based on GraphPlan [2]. With this planner it shares the same representation of disjunctive states, obtained by connecting facts and operators to form a graph, called *planning graph*.

DPPlan has, with respect to GraphPlan and other related planners, like IPP [11] or STAN, a completely different method for searching a solution in the planning graph.

The fundamental feature of DPPlan is that to each node of the graph a boolean value is assigned. An operator is *true* if it is executed, *false* if it is not. A fact is *true* if it is achieved by some operator, *false* otherwise. During the search phase a fact *p* can be *true* also if it is required by some operator *o* (*p* is a precondition of *o*) and *false* if it is required to be false (*p* is a negative precondition of *o*). This fact makes possible to use the propagation rules before the fact is really achieved and to cause, in the case, a backtracking earlier than it would be else obtained.

However those two situations are very different: if a fact *f* is *true* because something requires it, then *f* must be seen as a (sub-)goal and the current plan cannot be correct if this fact is not reached by any action. Only when an action achieving *f* is added to the plan, then *f* is really "*true*". The same distinction should be made between a fact which is really "*false*" (it has been deleted by some operator, or all of its achiever is false) and a fact required to be *false* (some action has it as a negative precondition).

In order to distinguish these situations, a further value, called *state*, is assigned to each fact:

- the state is "*produced*", when the fact value is reached (either true or false),
- the state is "*consumed*", if the fact value is required (either true or false), and
- the state is "*produced-after-consumed*" if the fact value has been required and then achieved.

This last value of state is useful during the backtracking phase.

In the previous paper of DPPlan we have described the rules by which it is possible to propagate a choice on the value of a node in the graph. These rules show how to update the other nodes of graph because of that choice. The operations of changing state and value of a node are named according to the type of changing and to the type of node they are applied.

The operation **use** sets the value of an operator node to *true*, while the operation **exclude** sets it to *false*.

The operation **consume** sets the value of a fact node to *true* and the state to *consumed*, **produce** sets the value to *true* and the state to *produced* (or *produced-after-consumed*), **consume-not** sets the value to *false* and the state to *consumed*, **destroy** sets the value to *false* and the state to *produced* (or *produced-after-consumed*).

All of these operations can fail if they try to give a different value to an already valued node (e.g. **destroy** and **consume** on the same fact) or can be ignored if they try to give the same (or compatible) value or state (e.g. **produce** and then **consume** on the same fact). For further details see the original paper [1].

For an operator *o*, *In(o), NotIn(o), Out(o)* and *NotOut(o)* are respectively the list of its positive preconditions, negative preconditions, positive effects and negative effects. For a fact *f*, *In(f), NotIn(f), Out(f)* and *NotOut(f)* are respectively the list of the operators which have *f* as a positive effect, as a negative effect, as a positive precondition and as a negative precondition; moreover *npp(f)* and *npd(f)* are respectively the number of possible producer (destroyer), i.e. the number of elements of list *In* and *NotIn* whose value is still undefined. Finally, for any node *n*, *Mutex(n)* is the list of the nodes exclusive of *n*.

These propagation rules update the state and the value to the node they are applied, as well as the goal list. Note that without any particular additional procedure, DPPlan is able to solve problem with negative preconditions and goals, and with temporally qualified "*initial states*", like the fact *f* is true at time $t > 0$, and goals, like the goal *g* has to be achieved at time $t < t_{max}$.

The main algorithm operates in way resembling the celebrate Davis-Putnam algorithm for propositional logic. At the beginning all the variables receive the value *undefined*, then the procedure *produce* is performed for all the facts in the initial state, *consume* is performed for all the positive goals, and *consume-not* is performed for all the negative goals.

In its main loop, the algorithm chooses an undefined variable *v*, tries to set its value to one of the boolean values (e.g. *true*) until it finds a solution (no fact is in the state *consumed*), or some propagation fails, in this case a backtracking phase is performed by undoing all the propagations done after *v* and tries to set the value *v* to the opposite value (e.g. *false*). If a failure is obtained again, *v* can be neither *true* nor *false*, therefore the backtracking stops until the previously tried variable is unset: now the value of this variable is reversed and the search goes on. Note that when the algorithm has tried both the value for the first chosen variable, without reaching a solution, the search phase is ended, the graph is augmented with all the new applicable operators and all the new facts they produce, in a way similar to the expansion phase of GraphPlan, and a new search phase is performed.

What is completely free in our algorithm is how to choose what variable to try next and which value to try first. According to the method used, the planner can perform a forward search, a backward-chain search, a bidirectional search or simply a non directional search.

In the original paper [1] we have listed several ways of choosing a variable, essentially an operator to be tried to *use*, and then to *exclude*.

RDDPlan: Planning with numerical resources in DPPlan

RDPPlan is the extension of DPPlan to handle numerical resources. In the model together with the logical propositions, the use of numerical variables is allowed. Each numerical variable, called *resource*, can be cited as in preconditions and goals as well as in effects and initial states. Conforming to the PDDL 2.1 [7], resources are represented as a numerical functions whose parameters can be domain constants or action parameters.

Preconditions and goals

In preconditions and goals it is allowed to use conditions like `(compare resource value)` where `compare` is a comparison operator (`>, <, >=, <=, =`) and `value` can be an expression involving action parameters, numerical constants and arithmetic operators.

Since we do not allow for disjunctive preconditions and goals, several constraints on the same resource *r* reduce to a unique real interval, possibly empty, indicated with $P_{A,r}$ for the precondition of action *A* and with G_r for goals. These intervals are possibly unlimited in left and/or in right side.

Initial state and effects

In the initial state the resources are initialized by the proposition `(= resource value)` where now the value can only be a numerical constant.

In the effects a resource can be changed by proposition `(change resource value)` where `change` can be one of the operators `assign, increase, decrease` and `value` is an expression as in the preconditions. We indicate the value added of action *A* to resource *r* with $E_{A,r}$, intending $E_{A,r}$ =*value* for *increase* and $E_{A,r}$ = -*value* for *decrease*. For an action *A* which does not change a resource *r* we treat *A* as an increase operator of value 0, i.e. $E_{A,r}=0$.

As a syntactic sugar we allow to use, while expressing preconditions and effects, in the expressions called *value* some other resources, provided they are static, i.e. initialized in the initial state, but not changed by any action. A static resource is a sort of constant which remains unchanged during the plan, like weight function in domain *Depots*.

This restriction has the main effect that each resource can be changed independently from the others. Allowing to cite other (non static) resources in the preconditions would have generated more complex admissible resource domains, not reducible to the cartesian product of real intervals. On the other hand if an effect on resource *r* could depend on the value of some other (non static) resource, some interaction between resources would have arisen which are difficult to handle (e.g. increasing a resource can cause to another resource to decrease).

Realized value and desired interval

Associated to each resource *r* and each time level *t*, a numerical value R_{rt} and a real interval D_{rt} is computed during the planning phase. Every time an action is selected by the search procedure, R_{rt} and D_{rt} are updated for every resource *r* used by the action and for every time level *t* greater or equal to the time level at which the action is selected. The previous values of R_{rt} and D_{rt} are restored during the backtracking phase.

The number R_{rt} represents the current value of the resource *r* at time *t* realized by the actions since now inserted in the plan. At the start of search procedure, for each resource *r* and for every time *t*, R_{rt} is set to the value specified at the initial state.

The interval D_{rt}, called "*desired interval*", contains all the admissible values for the resource *r* that allow the execution of all the actions selected at time level *t*. At the start of search procedure for each resource *r* and for every time $t < T$ (*T* is the last time level), D_{rt} is set to $[-\infty,+\infty]$, while for each resource *r*, D_{rT} is set to G_r, the interval specified by the goals.

Solution plans and executability

Definition 1 *The obvious sufficient and necessary condition for a plan to be executable and to be a solution of the given planning problem is that for each resource r and for each time level t the condition* $R_{rt} \in D_{rt}$ *holds.*

Before describing the rules with which R_{rt} and D_{rt} are updated, we must define when two or more actions are executable at the same time level. We use the same concept of simultaneous executability as expressed in [7,15].

Definition 2 *A set of actions* $A_1, A_2, \ldots, A_m$ *is simultaneously executable if for every permutation* Π *of the actions in the set* $A_{\Pi(1)}, A_{\Pi(2)}, \ldots, A_{\Pi(m)}$

1) $A_{\Pi(1)}$ *is executable in the current state,* $A_{\Pi(2)}$ *is executable in the state after the execution of* $A_{\Pi(1)}$*,* $A_{\Pi(3)}$ *is executable in the state after the execution of* $A_{\Pi(1)}$ *and then* $A_{\Pi(2)}$*, and so on,*

2) the effect over the resources is always the same.

As a straightforward consequence of the second condition, an assignment on resource *r* is not simultaneously executable with any action changing *r* (additive operator).

The question remains open whether to allow an action changing *r* to be simultaneous with any action having a precondition with respect to *r*. Our approach is to allow simultaneity whenever the change does not affect the executability.

Proposition 1 *It is easy to prove that two additive actions* A_1 *and* A_2 *are simultaneously executable (with respect to r) if, let* $[\alpha_1,\beta_1]=P_{A1,r}$ *and* $[\alpha_2,\beta_2]=P_{A2,r}$ *be respectively the precondition intervals and let* $k_1=E_{A1,r}$ *and* $k_2=E_{A2,r}$ *their effect on r, we have* $\alpha \le \beta$ *where* $\alpha=\max\{\alpha_1, \alpha_1-k_2, \alpha_2, \alpha_2-k_1\}$ *and* $\beta=\min\{\beta_1, \beta_1-k_2, \beta_2, \beta_2-k_1\}$*. In*

the positive case we set $D_{rt}=[\alpha, \beta]$ if actions A_1 and A_2 are the only actions to be executed at time t. Otherwise A_1 and A_2 are marked to be mutually exclusive.

The generalization to the case of many additive (over the same resource *r*) actions A_1, A_2, …, A_m is somehow straightforward.

Proposition 2 *Called $[\alpha_i,\beta_i]=P_{Ai,r}$ the interval precondition and $k_i=E_{Ai,r}$ their effects, for i=1,…,m, we have that the action A_i's are simultaneously executable (with respect to r) if $\alpha \leq \beta$ where $\alpha = \max\{\alpha_i - km_i : i=1,\ldots,m\}$, $\beta = \min\{\beta_i - kp_i : i=1,\ldots,m\}$ and $kp_i = \sum_{j\neq i, k_j>0} k_j$ and $km_i = \sum_{j\neq i, k_j<0} k_j$ for i=1,…,m.*

In the positive case $D_{rt}=[\alpha,\beta]$ if the actions A_i's are the only actions to be executed at time t..

α and β can be computed in linear time (in the number of resources and actions) by storing (and keeping updated) the values $kp_0 = \sum_{k_j>0} k_j$ and $km_0 = \sum_{k_j<0} k_j$.

The case of several additive actions includes the case of simultaneous execution of additive actions on resource *r* (possibly none) with other actions which do not change *r*. In the particular case, where all the actions do not change *r*, D_{rt} reduces to the intersection of all the precondition intervals.

A case not yet covered is the simultaneous execution of an assignment on *r*, say A_1 whose assigns to *r* the value *v*, with with actions $A_2,\ldots,A_m$ which do not change *r*. It is easy to see that $A_1,A_2,\ldots,A_m$ are simultaneous executable (with respect to *r*) if the intersection of all the precondition intervals is not empty and contains *v*.

After a new action *A* is selected to be *used* at time *t*, we must check if it is simultaneously executable with the already selected actions at time *t*, by computing for each resource *r* the quantities $\alpha(r)$ and $\beta(r)$. Only when each of these interval is not empty, then the desired interval for *r* is set to be $[\alpha(r), \beta(r)]$. Moreover an incremental way, which takes only a constant time to be computed, of updating D_{rt}, after the *use* of a new additive operator *A*, is the following.

Proposition 3 *If $D_{rt} = [\alpha,\beta]$, $P_{A,r} = [a,b]$ and $E_{A,r} = k$, then the updated desired interval is $[\alpha',\beta']$ where*

$\alpha' = \max\{a - km_0, \alpha - \min\{k,0\}\}$ and $\beta' = \min\{b - kp_0, \beta - \max\{k,0\}\}$.

Updating the values R_{rt} is done by recomputing them for every resource changed by *A*, starting from time *t+1* and ending at the first time where an assignment over *r* is selected. This computation can be efficiently done by storing and keeping updated the total amount to be added to *r*, computed considering all the actions selected since now.

Extension to interval resources

A very straightforward, yet significant, extension of the simple model above explained is to allow for non completely specified initial states and effects.

Interval on the initial states and effects

Instead of initializing a resource with a unique real value, we allow to specify a real interval I_r as a range for the initial value of the resource *r*. The planner operates in an under-specified domain in which the value of some resource is not exactly known, but it is bound to be in an interval. Suppose we do not know exactly how much gasoline is in the tank of our car: we just know that it surely the real amount is between 5 and 10 liters.

Similarly it is possible to have under-specified effects of any operator: the value which is added, subtracted or assigned to the current value of a resource is not exactly known, but only a lower and an upper bound is specified. Imagine that the car in the previous example, we do not know which is the exact consumption: all we know is that the car can travel from 10 to 15 kilometers per liter.

In this enhanced model, the real quantities $E_{A,r}$ and R_{rt} are therefore replaced by real intervals, which cannot be unlimited in the left or in the right side. The intervals R_{rt} are initialized with I_r, the intervals specified in the initial state description, and are updated according the following simple rules, where the current interval for R_{rt} is $[\gamma,\delta]$, the operator to be executed is *A* and $E_{A,r} = [e_{min}, e_{max}]$: if *A* is `ASSIGN` R_{rt} becomes $[e_{min}, e_{max}]$, if *A* is `INCREASE` R_{rt} becomes $[\gamma + e_{min}, \delta + e_{max}]$ and if *A* is `DECREASE` R_{rt} becomes $[\gamma - e_{max}, \delta - e_{min}]$.

Solution plans and executability

The definition of what solution plan is meant is similar to the definition described in the previous section.

Definition 3 *For this model the necessary and sufficient condition for a plan to be a solution of a given planning problem is that for each resource r and time level t, $R_{rt} \subseteq D_{rt}$.*

The intended semantics of this meaning of the term *solution plan* Π is that if for every possible way of replacing each interval effect $E_{A,r}$ with a number $e_{A,r} \in E_{A,r}$ and of replacing each initial interval I_r with a number $i_r \in I_r$, the problem so obtained, which now is conform to the previous model, is solved by Π, according to the semantics expressed in the previous section. Expressed in other terms, a solution plan must solve every possible problem that is allowed by the constraints specified in the initial state and in the effects description.

The update rules for desired interval are similar to what we have seen in the previous section.

Proposition 4 *Suppose that A_1 and A_2 be two additive actions with precondition intervals $P_{A1,r}=[\alpha_1,\beta_1]$ and $P_{A2,r}=[\alpha_2,\beta_2]$ and with effect intervals $E_{A1,r}=[e_{min,1},e_{max,1}]$ and $E_{A2,r}=[e_{min,2},e_{max,2}]$ respectively. If $\alpha \le \beta$, where $\alpha = max\{\alpha_1, \alpha_1-e_{min,2}, \alpha_2, \alpha_2-e_{min,1}\}$ and $\beta = min\{\beta_1, \beta_1-e_{max,2}, \beta_2, \beta_2-e_{max,1}\}$, then A_1,A_2 are simultaneously executable.*

The generalization to the cases with m additive or multiplicative actions are the following.

Proposition 5 Called $P_{Ai,r}=[\alpha_i, \beta_i]$ the precondition intervals, for i=1,…,m and $E_{Ai,r}=[e_{min,i},e_{max,i}]$ the effect intervals over r for i=1,…,m, we have that the action A_i's are simultaneously executable if $\alpha \le \beta$, *where* $\alpha = max\{\alpha_i - km_i : i=1,\dots,m\}$, $\beta = min\{\beta_i - kp_i : i=1,\dots,m\}$ and $kp_i = \sum_{j\neq i, e_{\max,j}>0} e_{\max,j}$ and $km_i = \sum_{j\neq i, e_{\min,j}<0} e_{\min,j}$.

Strategies on Resources

In this section we present the strategies that we have defined for achieving the goals over resources. These strategies are necessary to solve "*pure numerical problems*", i.e. problems with goals only on resources. The methods that implement these strategies are combined with the ones for solving logical goals, by evaluating the difficulties of resources and logical goals and selecting the most difficult goal to solve.

Strategy for numerical resources

For each time step *t* and for each resource *r*, we check if the condition $R_{rt} \in D_{rt}$ holds: the negative cases are the goals on resources. An action that can help solving a goal on resource *r* at time *t* can be chosen according to the following rules.

First, the algorithm searches for an action that can achieve the goal in one only step, preferring, in the case of many available options, the action situated at the level nearest to *t*. If such an action does not exist, we choose an increaser (actions which make R_{rt} bigger) or a decreaser (actions which make R_{rt} smaller) according to the case. In the detail, a first search is performed for all those actions *A* present at any time $\tau < t$ such that the updated value (if *A* would be executed) of R_{rt}, say R'_{rt}, verifies the condition $R'_{rt} \in D_{rt}$. Among those, the action *A* with the highest τ is selected, by performing the search starting from the time-step *t–1* and going backward: the first action found is chosen.

If the first search fails, the algorithm chooses an action that can permit us to come closer to the goal. Called $[\alpha,\beta]$ the interval D_{rt}, the algorithm selects an increaser *A*, if $R_{rt} < \alpha$, or a decreaser *A*, if $R_{rt} > \beta$, that minimizes $\min\{|\alpha - R'_{rt}|, |R'_{rt} - \beta|\}$.

Strategy for interval resources

When we work with resources as intervals using the "interval algebra" explained in a previous section, we have to handle with real intervals whose width can in general only grow, except when an assignment is performed. In fact it is obvious that any numerical operation between intervals produce as a result an interval which has a width larger than the original widths.

The following example can show this characteristic. If we have in the tank an amount of fuel that we do not know

exactly, but that we know be between *10* and *15* liters (this is a case of incomplete knowledge in initial state) and we take from an another tank an unknown amount of fuel between *12* and *16* liters (example of non deterministic effects over resource), then the minimum amount of fuel in the first tank is *22* liters and the maximum is *31* liters. So, in the notations here used, from the rule $R_{rt} + E_{A,r} = R_{r,t+1}$, we will obtain *[10,15] + [12,16] = [22,31]*. This means that we have at time step *t*, before the action application, a realized interval with width *5* and then, at the next time-step *t +1*, an interval with width *9*.

If you think that the width of realized interval represents, in some sense, the indetermination on resource value, we have that the larger the interval width, the larger the indetermination. Moreover note that if the width of the realized interval is large, it is more difficult that the solution conditions $R_{rt} \subset D_{rt}$ will hold. Let |*R*| denote

the width of interval *R=[a,b]*, i.e. *|R|=b-a*. The first control to do is on the widths of realized and desired intervals.

1) If $|R_{rt}| > |D_{rt}|$ an assignment which assigns an interval with width less than $|D_{rt}|$ is the only possible choice. If there are many such assignments, the algorithm chooses that one which assigns the interval with the least width. If there are no assignments with this property, a backtracking is necessary.
2) If $|R_{rt}| \leq |D_{rt}|$ the algorithm tries to solve this goal using a procedure similar to that described in the previous section. Now the choice criteria takes into account the distance between D_{rt} and R'_{rt} (which is the realized interval updated after the execution of the action to be evaluated) and their widths.

The previously described criteria can be implemented by defining two preference functions, one for the interval widths, and one for the distance between intervals, and by searching for an action that maximizes a linear combination of the functions. The first function is $f_W(A, D_{rt}) = \frac{|D_{rt}| - |R'_{rt}|}{|D_{rt}|}$ and gives to each action *A* a numerical positive score between 0 and 1.

The second function is a decreasing function of the distance between the middle points of the two intervals $f_D(A, D_{rt}) = \exp(-\frac{1}{2}|\gamma + \delta - \alpha - \beta|)$ where $[\alpha,\beta] = D_{rt}$ and $[\gamma,\delta] = R'_{rt}$.

Also this function gives to each action *A* a numerical positive score between 0 and 1.

Experimental results show that good values for the ratio c_1/c_2 of the coefficients of the linear combination $f(A, D_{rt}) = c_1 f_W(A, D_{rt}) + c_2 f_D(A, D_{rt})$ are between 1 and 2.

A theoretical justification is that f_W is slightly more influent for driving the search algorithm, because it can be useless to get closer to the desired interval if the width of the realized interval is too large.

Conclusion

RDDPlan a model of planning with interval resources based on propagation rules has been described. This models seems to be more adequate than existing models of planning in order to describe real world operators which uses resources because it does not require a complete knowledge of the quantity to be updated, but an interval boundary.

The main contribution to RDPPlan comes from [1], whose propagation and failure rules have been extended with interval management. Other related works share with it a similar planning graph structure with a different semantics for resources and no management of intervals [12,15]. Although an actual planner and strategies for resources have been implemented on the basis of the proposed model, RDPPlan can be considered a platform on which strategies and heuristics for planning with resources can be experimented. Further investigations and experiments are planned in order to develop more accurate heuristics and strategies which takes into account of resources, moreover, in order to provide a meaningful evaluation it will be also required the development of a set of significant benchmarks for planning domains with interval resources.

Finally it is worth investigating further extensions to the resources model more accurate with respect to the uncertainty in the real world e.g. intervals with given probability distribution over resources values and fuzzy quantities.

Bibliography

[1] M.Baioletti,S.Marcugini,A.Milani. DPPlan: an algorithm for fast solution extraction from a planning graph. *In Proc. of AIPS-00*, 2000.

[2] A.Blum,M.Furst. Fast planning through planning graph analisys. *Artificial Intelligence (90):281-300*, 1997.

[3] B.Bonet,H.Geffner. Planning as heuristic search. *Artificial Intelligence 129*, 2001.

[4] S.Chien et al. ASPEN automated planning and scheduling for space mission operation. *In Proc of SpaceOps2000*, 2000.

[5] K.Currie,A.Tate. O-Plan: The open planning architecture. *Artificial Intelligence (52):49-86*, 1991.

[6] M.Do,S.Kambhampati. Sapa: A domain indipendent heuristic metric temporal planner. *In Proc. of ECP-01*, 2001

[7] M.Fox,D.Long. PDDL 2.1: An extension to PDDL for expressing temporal planning domains. Forthcoming in JAIR special issue on 3rd International Planning Competition

[8] J.Hoffmann,B.Nebel. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research (14): 253-302*, 2001.

[9] J.Hoffmann. Extending FF to numeric state variables. *In Proc. of ECAI-02*, 2002.

[10] A.Jonsson et al. Planning in the interplanetary space : theory and practice. *In Proc. of AIPS-00*, 2000.

[11] J.Koehler,B.Nebel,J.Hoffmann,Y.Dimopoulos. Extending planning graphs to an ADL subset. *In Proc. of ECP-97*, 1997.

[12] J.Koehler. Planning under resource constraints. *In Proc. of ECAI-98*, 1998.

[13] P.Laborie,M.Ghallab. Planning with sharable resource constraints. *In Proc of IJCAI-95*, 1995.

[14] X.Nguyen,S.Kambhampati,R.S.Nigenda. Planning graph as the basis for deriving heuristics for plan synthesis by state space and csp search. *ASU Technical Report*, 2002.

[15] J.Rintanen,H.Jungholt. Numeric state variables in constraint based planning. *In Proc. of ECP-99*, 1999.

[16] S.Wolfman,D.Weld. The LPSAT engine and its application to resource planning. *In Proc. of IJCAI-99*, 1999.

Authors information

Marco Baioletti – Dipartimento di Metodi Quantitativi, Univeristà degli Studi di Siena, P.zza S.Francesco 5, Siena, Italy

Alfredo Milani - Dipartmento di Matematica e Informatica, Università degli studi di Perugia, Via Vanvitelli, 06100 Perugia, Italy

Valentina Poggioni – Dipartimento di informatica e Automazione, Università degli Studi di Roma Tre, Via della Vasca Navale 79, 00146 Roma, Italy

A PLANNING MODEL WITH RESOURCES IN E-LEARNING

G. Totkov, E. Somova

Abstract: This work proposes a model for planning of education based on resources and layers. Each learning material or concept is determined by certain characteristics: a layer and a list of resources and resource values. Models of studied subject domain, learner, information and verification unit, learning material, plan of education and education have been defined. The plan of education can be conventional, statical, author's and dynamic. Algorithms for course generation, dynamic plan generation and carrying out education are presented. The proposed model for planning of education based on resources and layers has been included in the system PeU.

Keywords: planning education, e-learning.

Introduction

The e-learning action plan has been created by the Commission of the European Communities in 2001. The development of this plan shows the actuality of the present work. A lot of attempts for creation the e-learning standards with necessary requirement (SCORM, CMI, LOM, IMS, ARIADNE) and systems for creation of e-learning environments (Learning Space, WebCT, Top Class, First Class, Blackboard, Virtual-U, Web course in a Box, CourseInfo, Learning Landscapes, CoSE, CoMentor, ARIADNE, Asymetrix Librarian, Norton Connect, Allaire Forum, Team Wave, WebBoard, Asymetrix ToolBook, etc.) [Britain] have been done. In order to satisfy the requirements of the e-learning environments, the model of planning of education have to be done.

The known models for planning of education are classical [Koffman, 1975; Пасхин, 1985; Савельев, 1986; Зайцева, 1989; Grandbastien, 1994; Grant, 1997] and resource [Milani, 2000; Milani, 2001a; Milani, 2001b] respectively based on the classical and recourse models of problem planning.

The classical model for planning [Chien] is based on the initial and the final state of the problem and the operators for transformation of one state into another. The Chien model can not represent multiple used educational environment, where learners are interested in optimizing the path for passing over learning materials, not only from the point of view of the length of the path, but also depending on the time for passing over, price, level of difficulty, etc.