A Planning Approach to Deal with Numeric Variables in Multiobjective Planning

Antonio Garrido[†] and Derek Long[‡]

[†] Universidad Politécnica de Valencia, Valencia (Spain)
[‡] University of Strathclyde, Glasgow (UK)
agarridot@dsic.upv.es derek.long@cis.strath.ac.uk

Abstract. Planning problems have been traditionally expressed by a propositional representation. However, this representation is not adequate to express numeric variables when modeling real-world continuous resources, such as fuel consumption, energy level, profit, etc. This paper presents our ongoing work to build a planner with capabilities for numeric variables, including duration on actions, and multiobjective optimisation to manage level 3 durative actions of PDDL2.1. Our approach consists of two stages. First, a spike construction process estimates the values of the variables associated to propositions/actions. Unlike other approaches, we do not relax numeric effects in the calculus of the estimation, but only numeric conditions. Second, a heuristic search process generates a relaxed plan according to the estimations of the first stage, and then performs search in a plan space. The relaxed plan and the heuristic estimations help the process find a plan while trying to optimise the multiobjective criterion.

1 Introduction

Planning problems have been traditionally expressed by means of a purely propositional representation based on STRIPS [1] and its successors [2, 3]. However, this representation is not always enough to express real-world problems, particularly in problems which involve time, i.e. actions with duration, and continuous resources [4, 5]. In a propositional representation, Graphplan's planning graph [6] has become tremendously useful to estimate when propositions/actions are given [7–9]. Furthermore, this graph can be generated in polynomial complexity (both in time and space) under such a representation, where propositions/actions have binary domains (true or false).

Nevertheless, when dealing with time explicitly stored in the graph levels, the planning graph becomes more complex. Now, the levels are not longer equidistant and the number of levels is significantly increased [10]. More complex is indeed the fact of dealing with numeric variables, such as fuel consumption, energy level, pollution level, distances, profit, etc., where the domain of the variables is usually continuous, i.e. formed by values in $\mathbb{R}$. In this case, a traditional planning graph cannot be directly used because the conditions of the actions may impose complex constraints (inequalities) on these variables, such as `(fuel plane)` $\geq$ `100`, `(available-space truck)` $>$ `50`, `(distance)` $\leq$ `20`, etc. Analogously, effects of the actions may modify these values in many ways, such as `increase (fuel plane) 10`, `decrease (available-space truck) 5`, `scale-up (profit) 1.5`, etc.

Some researchers have exploited the argument of relaxing the planning graph by ignoring the delete effects to heuristically estimate and manage numeric variables. For instance, in GRT-R [11], the resource consumption of the actions is used as an additional cost of the heuristic function. In MO-GRT [5], numeric variables are used to cope with multiobjetive heuristic planning. Sapa [12] and TP4 [13] use the resource consumption to assess the necessary increment in the heuristic value that estimates the length of the plan. Unlike these planners, metric-FF [4, 14] also considers the numeric conditions of the actions, but it still ignores the decreasing effects in the heuristic estimation. Consequently, in many cases these *effect relaxing* heuristics represent very optimistic estimations, far away from the real values.

In this paper, we present our ongoing work to build a planner with capabilities for numeric variables, including duration on actions, and multiobjective optimisation to manage level 3 durative actions of PDDL2.1 [15]. Level 3 of PDDL2.1 allows a combination of logical (propositional) and numeric features on actions, which have local conditions and effects. This work extends and stresses on the initial ideas for heuristic search in PDDL2.1 temporal planning of TPSYS [10, 16], and attempts to solve its main inefficiencies and limitations.

The paper is structured as follows. Section 2 reviews the foundations of TPSYS and indicates its main limitations. Section 3 introduces the preliminaries about expressing numeric variables in action conditions and effects. Our approach to deal with numeric variables is presented in section 4. Section 5 discusses some points about the management of numeric variables in conditions to improve the heuristic estimations. Finally, section 6 presents the conclusions and related work.

2 A Brief Review of TPSYS

TPSYS is a temporal planner that manages the non-conservative model of durative actions of PDDL2.1 and consists of three stages as depicted in Figure 1 [16]. Like other Graphplan-based planners, TPSYS identifies binary mutual exclusion relations between actions and between propositions. Local conditions and effects of PDDL2.1 actions make the mutex calculus more complex because there exist different ways of overlapping actions. Therefore, after instantiating the actions, the first stage calculates the action/action and proposition/action *static* mutex relations. These mutex are static because they only depend on the definition of the actions and, consequently, always hold. Although this calculus speeds up the remaining stages, it entails an important **inefficiency**. In problems with hundreds or thousands of actions, a lot of effort is wasted calculating the mutex between actions which in the plan never interact.

The second stage incrementally extends a temporal planning graph, alternating proposition and action levels, and calculates the action/action, proposition/action and proposition/proposition dynamic mutex. As in the first stage, a lot of effort is wasted in the mutex calculus between actions which will not interact in the plan. Moreover, the different duration of the actions breaks the original symmetry of the temporal graph and the levels are not equidistant. This entails a new indication of **inefficiency**. In problems where the greatest common divisor of the durations is 1, the algorithm must generate many levels until all the problem goals are non-pairwise mutex, thus increasing the size of the graph, the complexity of the second stage and, subsequently, of the search.

The third stage performs the search of a temporal plan. This stage is divided into two new stages. First, a relaxed plan (without taking any mutex into consideration) is generated in a backward chaining way. Second, this relaxed plan is used as a *skeleton* to generate the temporal plan and as the basis for calculating heuristic estimations in a forward chaining way. This stage may extend the temporal graph as much as necessary until finding a plan, with the cost associated to the mutex calculus (see Figure 1). In addition, the search in the third stage entails a **limitation**: plans are optimised only *w.r.t.* makespan. No other criteria, such as resource consumption or action cost can be used, which limits the capabilities of the planner to deal with more general real problems.

3 Preliminaries. Numeric variables in actions

This section introduces the notation for the numeric variables in actions used throughout the rest of the paper. We use durative actions as defined in PDDL2.1, thus presenting three types of conditions: i) $SCond(\texttt{a})$, with the conditions to be guaranteed at the start of `a`; ii) $Inv(\texttt{a})$, with the invariant conditions to be guaranteed over the execution of `a`; and iii) $ECond(\texttt{a})$, with the conditions to be guaranteed at the end of `a`. In a numeric approach, each condition can be a *classical* propositional condition or a numeric condition that expresses a constraint as a tuple $\langle$`f-exp1,binary-comp,f-exp2`$\rangle$, where `f-exp1` and `f-exp2` represent functional expressions (which may contain numeric variables), and `binary-comp`

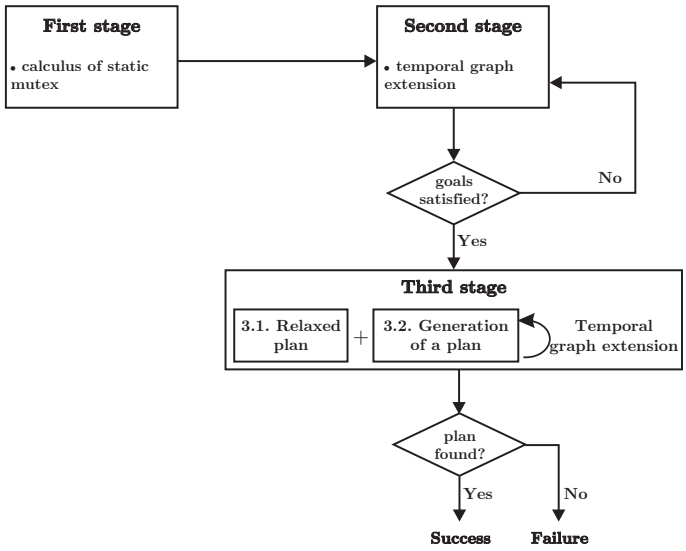

Fig. 1. Structure of TPSYS.

$\in \{<, \leq, =, >, \geq\}$ is a binary comparator. Additionally, durative actions have two types of effects: i) $SEff(\mathtt{a})$, with the effects to be asserted at the start of `a`; and ii) $EEff(\mathtt{a})$, with the effects to be asserted at the end of `a`. Now, each effect can be either a positive $(SAdd(\mathtt{a}), EAdd(\mathtt{a}))$ or negative $(SDel(\mathtt{a}), EDel(\mathtt{a}))$ propositional effect or a numeric effect as a tuple $\langle$`f-head,assign-op,f-exp`$\rangle$, where `f-head` represents a numeric variable, and `assign-op` $\in \{:=, +=, -=, *=, /=\}$ is an assignment operator which updates the value of `f-head` according to the functional expression `f-exp`.

Figure 2 shows the `zenotravel` domain with numeric conditions and effects used in the last IPC–2002[1]. For instance, action `fly` requires the plane to be in the origin (propositional condition), but also to have enough fuel to carry out the flight (numeric condition). On the other hand, this action makes the plane to be in the destination (propositional effect) and also increases the variable `total-fuel-used` and decreases the `level` of fuel of the plane (numeric effects). By default, all the domains include the variable `(total-time)`, which represents the makespan of the plan, and that all the actions modify by the field `:duration`. This way, the duration is simply treated as one of the numeric variables. Clearly, different actions may present different numeric effects: they can be fixed for all actions of an operator, different for each action of an operator (depending on its arguments), or even different for the same action (depending on the state where they are applied). Let us consider for example the case of the variable `(total-time)`. Any action of type `board` always increases the same value `boarding-time`; a particular action `fly` always increases the value `(/ (distance ?c1 ?c2) (slow-speed ?a))`, whereas a particular action `refuel` increases the value `(/ (- (capacity ?a) (fuel ?a)) (refuel-rate ?a))`, which depends on the current value of the variable `(fuel ?a)`.

Although in this domain there exist many numeric variables (`distance`, `slow-speed`, `fuel`, etc.) many of them are *static* (never affected by actions) and, consequently, not considered during planning. For instance, if we assume that there exists only one plane `plane1`, the numeric variables are `(total-fuel-used)`, `(fuel plane1)` and `(total-time)`. In addition to this, each problem of the domain can define a multiobjective metric function to assess the quality of the plan, such as `(+ (* 4`

[1] More information on this and other domains of the International Planning Competition 2002 in: `http://www.dur.ac.uk/d.p.long/IPC`.

```
(:durative-action board
 :parameters (?p - person ?a - aircraft ?c - city)
 :duration (= ?duration (boarding-time))
 :condition (and (at start (at ?p ?c))
                 (over all (at ?a ?c)))
 :effect (and (at start (not (at ?p ?c)))
              (at end (in ?p ?a))))

(:durative-action debark
 :parameters (?p - person ?a - aircraft ?c - city)
 :duration (= ?duration (debarking-time))
 :condition (and (at start (in ?p ?a))
                 (over all (at ?a ?c)))
 :effect (and (at start (not (in ?p ?a)))
              (at end (at ?p ?c))))

(:durative-action fly
 :parameters (?a - aircraft ?c1 ?c2 - city)
 :duration (= ?duration (/ (distance ?c1 ?c2) (slow-speed ?a)))
 :condition (and (at start (at ?a ?c1))
                 (at start (>= (fuel ?a) (* (distance ?c1 ?c2) (slow-burn ?a)))))
 :effect (and (at start (not (at ?a ?c1)))
              (at end (at ?a ?c2))
              (at end (increase total-fuel-used (* (distance ?c1 ?c2) (slow-burn ?a))))
              (at end (decrease (fuel ?a) (* (distance ?c1 ?c2) (slow-burn ?a))))))

(:durative-action zoom
 :parameters (?a - aircraft ?c1 ?c2 - city)
 :duration (= ?duration (/ (distance ?c1 ?c2) (fast-speed ?a)))
 :condition (and (at start (at ?a ?c1))
                 (at start (>= (fuel ?a) (* (distance ?c1 ?c2) (fast-burn ?a)))))
 :effect (and (at start (not (at ?a ?c1)))
              (at end (at ?a ?c2))
              (at end (increase total-fuel-used (* (distance ?c1 ?c2) (fast-burn ?a))))
              (at end (decrease (fuel ?a) (* (distance ?c1 ?c2) (fast-burn ?a))))))

(:durative-action refuel
 :parameters (?a - aircraft ?c - city)
 :duration (= ?duration (/ (- (capacity ?a) (fuel ?a)) (refuel-rate ?a)))
 :condition (and (at start (> (capacity ?a) (fuel ?a)))
                 (over all (at ?a ?c)))
 :effect (at end (assign (fuel ?a) (capacity ?a))))
```

Fig. 2. `Zenotravel` domain with numeric capabilities used in the *Time* track in IPC–2002.

(total-time)) (* 0.005 (total-fuel-used))), which can either be minimised or maximised. Unlike planners that only try to optimise the number of planning steps, actions or makespan, the use of this metric allows to find plans where several weighted criteria that play an important role in the plan are also considered.

4 Planning with numeric variables

In the new planning approach, the original first and second stage of TPSYS are merged into one only stage. Unlike TPSYS, we do not generate a real temporal planning graph because the levels do not play any role now. This allows to overcome two of the most important inefficiencies of TPSYS: i) the calculus and propagation of many mutex, and ii) the expensive extension of a temporal planning graph. Instead of extending a temporal graph, we generate one spike vector for propositions and one for actions, as in STAN [17]. Consequently, the first stage performs both the instantiation of the actions through the spike construction and the heuristic estimation of the numeric variables. Next, the second stage performs the heuristic search of a plan, which is assessed in terms of the multiobjective metric function, thus increasing the capabilities of TPSYS.

4.1 First stage. Spike construction and estimation of the numeric variables

The first stage uses two spikes that encode the information about propositions/actions, traditionally stored in a planning graph, thus reducing the storage requirements. Each proposition/action is associated with a vector of tuples $\langle \texttt{f-head}_i, \texttt{min\_value}_i, \texttt{max\_value}_i \rangle$, where $\texttt{f-head}_i$ is the ith numeric variable and $\texttt{min\_value}_i$ ($\texttt{max\_value}_i$) stands for the minimal (maximal) value estimated for that variable to achieve the proposition/action. These values are incrementally updated through the spike construction as shown in Algorithm 1.

1: {*Initialising propositions and the values of numeric variables present in the initial state* $\mathcal{I}_s$}
2: **for all** $\texttt{p} \in \mathcal{I}_s$ **do**
3: $\quad$ *propositional_state* $\leftarrow$ *propositional_state* $\cup \{\texttt{p}\}$
4: $\quad \forall \texttt{f-head}_{\texttt{p},i}$, update $\langle \texttt{f-head}_{\texttt{p},i}$, value of $\texttt{f-head}_i$ in $\mathcal{I}_s$, value of $\texttt{f-head}_i$ in $\mathcal{I}_s \rangle$
5: {*Spike construction*}
6: **while** new actions can be applied from *propositional_state* **do**
7: $\quad$ **for all** $\texttt{a} \mid \{SCond(\texttt{a}) \cup Inv(\texttt{a})\} \in$ *propositional_state* **do**
8: $\quad\quad$ insert a in the spike of actions
9: $\quad\quad$ {*Estimating the values of numeric variables for the actions that can start*}
10: $\quad\quad \forall \texttt{f-head}_{\texttt{a},i}$, update $\langle \texttt{f-head}_{\texttt{a},i}, \cup^{T}_{\forall \texttt{p}_\texttt{j} \in SCond(\texttt{a}) \cup Inv(\texttt{a})} \texttt{min\_value(f-head}_{\texttt{p}_\texttt{j},i}\texttt{)}, \cup^{T}_{\forall \texttt{p}_\texttt{j} \in SCond(\texttt{a}) \cup Inv(\texttt{a})} \texttt{max\_value(f-head}_{\texttt{p}_\texttt{j},i}\texttt{)} \rangle$
11: $\quad\quad$ **for all** $\texttt{p} \in \{SAdd(\texttt{a}) \cup EAdd(\texttt{a})\}$ **do**
12: $\quad\quad\quad$ insert p in the spike of propositions (if not present)
13: $\quad\quad\quad$ *propositional_state* $\leftarrow$ *propositional_state* $\cup \{\texttt{p}\}$
14: $\quad\quad\quad$ {*Estimating the values of numeric variables for the effects*}
15: $\quad\quad\quad \forall \texttt{f-head}_{\texttt{p},i}$, update $\langle \texttt{f-head}_{\texttt{p},i}, \cap^{T}_{\min_{\forall \texttt{a}_\texttt{j} \text{ supporting p}}} \texttt{min\_effect(f-head}_{\texttt{a}_\texttt{j},i}\texttt{)}, \cap^{T}_{\max_{\forall \texttt{a}_\texttt{j} \text{ supporting p}}} \texttt{max\_effect(f-head}_{\texttt{a}_\texttt{j},i}\texttt{)} \rangle$

Algorithm 1: Spike construction and estimation of the numeric variables.

For each initial proposition p, the algorithm initialises the numeric variables $\texttt{f-head}_{\texttt{p},i}$ with the values indicated in the initial state $\mathcal{I}_s$ (steps 1–4). If one variable is not initialised in $\mathcal{I}_s$ (for instance, (total-time)), it is initialised to 0. Steps 5–15 construct the spike structure until no new actions can

be generated. For each action `a` that starts, step 10 estimates the minimal and maximal values of the numeric variables as the *temporal union* ($\cup^T$) of the values of the (start and invariant) conditions of `a`. The operation *temporal union* must take into account the duality in the optimisation direction of the variables. Consequently, $\cup^T$ is defined as the operation "max" when the variable must be minimised, and "min" when maximised. For instance, when dealing with the variable `(total-time)`, the estimation of that variable for `a` is the maximal value of its conditions (informally, `a` must *wait* until the latest of its conditions). On the contrary, the estimation for the variable `(fuel plane1)`, which represents the current fuel level of `plane1`, is the minimal value because `a` must *wait* until the condition with the lowest fuel level. Analogously, step 15 estimates the minimal and maximal values of the variables for propositions as the *temporal intersection* ($\cap^T_{\min}$ or $\cap^T_{\max}$) of the effects of their supporting actions (after applying the assignment operators $\{:=, +=, -=, *=, /=\}$). This operation is interpreted as "min" for the minimal value of the variable and "max" for its maximal value. This way, each numeric variable has always associated two values that correspond with the most optimistic and pessimistic estimations.

It is important to note a special characteristic during the estimation of the numeric variables: the numeric conditions of actions are relaxed, but not their effects as in other approaches [4, 12–14]. The heuristic estimation on actions is only calculated in terms of their propositional conditions. Hence, the estimation makes a clear distinction between the logic of the plan and the constraints on the resources. In the case of the `zenotravel` domain, the heuristic estimates the cost of the actions necessary to deliver the set of people and not the requirements on resources (`fuel ?a`) of these actions. Intuitively, the heuristic informs about the cost of the actions that should be executed to achieve the propositional problem goals, but not about the cost of the actions which achieve the appropriate conditions of the resources to execute them (i.e. the heuristic estimates the cost of actions `board`, `debark`, `fly` and `zoom`, but not `refuel`).

4.2 Second stage. Search of a plan

This stage is inspired by the original search of TPSYS (see Figure 1), and it is divided into two stages. First, a backward chaining stage generates an initial relaxed plan. Second, a forward chaining stage allocates the execution time of the actions in the relaxed plan trying to optimise the multiobjective optimisation function.

Generation of an initial relaxed plan A relaxed plan Π is a partially ordered set of actions connected by causal links in which both the propositional problem goals and action conditions hold. It is called relaxed because neither mutex relationships between actions nor commitment on their execution time are considered. Algorithm 2 shows the way to generate this plan.

1: $\Pi \leftarrow \{\mathtt{IS} \cup \mathtt{FS}\}$ {*obligatory actions*}
2: *queue_acts* $\leftarrow$ `FS`
3: **while** *queue_acts* $\neq \emptyset$ **do**
4: extract `a` from *queue_acts*
5: **for all** `p` $\in \{SCond(\mathtt{a}) \cup Inv(\mathtt{a}) \cup ECond(\mathtt{a})\}$ **do**
6: **if** `p` is not supported in Π **then**
7: $\mathtt{b} \leftarrow \arg\min_{\forall \mathtt{b_i} \text{ which supports } \mathtt{p}}$(cost of execution of $\mathtt{b_i}$ evaluated in the optimisation function)
8: **if** `b` is the only action which supports `p` $\wedge$ `a` is an obligatory action **then**
9: mark `b` as obligatory in Π
10: $\Pi \leftarrow \Pi \cup \{\mathtt{b}\}$ {*no commitment on execution time of* `b` *yet*}
11: *queue_acts* $\leftarrow$ *queue_acts* $\cup \{\mathtt{b}\}$

Algorithm 2: Generation of an initial relaxed plan Π.

The algorithm is quite straightforward and consists of making actions applicable by supporting their conditions. All the relaxed plans contain two fictitious actions with no duration called `IS` and `FS`. `IS` achieves the propositions of the initial state, whereas `FS` requires the problem goals. The algorithm uses a queue of actions (*queue_acts*), which is initialised with `FS` (step 2). Step 4 extracts one action `a` from *queue_acts*. For each unsupported condition of `a`, step 7 selects the action `b` with the minimal cost of execution[2]. This cost is calculated by evaluating the numeric variables (estimated in the first stage) in the optimisation function. We use the idea of *obligatory action* [16] to indicate that such an action must be present in all the plans because it is the only way to support the problem goals (steps 8–9). Finally, steps 10 and 11 insert action `b` into Π and *queue_acts*, respectively.

The actions in the relaxed plan are evaluated and selected according to the estimation of their numeric variables. Therefore, the relaxed plan aims at the logical part of the plan, i.e. the actions to achieve the propositional goals, and not at the actions to achieve the required values of the resources. Apparently, this distinction seems intelligent when dealing with large problems: the relaxed plan focuses on the general structure of the plan, without taking into consideration the actions to replace the resources, which in many cases might be unknown or irrelevant [12] in advance.

Generation of a plan. Planning and allocating actions This stage performs a plan space search in a structure called *set_of_plans*, which contains all the generated plans $\{\Pi_i\}$. Actions in each Π_i are divided into two disjunctive sets: $Relax_i$ and $Alloc_i$. $Relax_i$ contains the actions which have not been allocated yet, and so they can be removed from Π_i. $Alloc_i$ contains the actions which have been allocated in time and will never be removed from Π_i. Initially, $Relax_i$ contains all the actions in Π_i (the initial Π_i is the plan computed by Algorithm 2) and $Alloc_i$ is empty. Each plan Π_i also contains a stack of actions $acts\_to\_allocate_i$ with the actions to allocate in each $time\_of\_execution_i$. $acts\_to\_allocate_i$ is initially empty and $time\_of\_execution_i$ is initialised to 0.

The idea of this stage is to move forward in time, simulating the real execution of Π_i, progressively taking care of the actions which can start at the current state (see Algorithm 3). The algorithm extracts the plan Π_i of lowest cost from *set_of_plans* (step 3). If the problem goals are supported in $Alloc_i$ the algorithm terminates with success (steps 4–5). If any action in $Alloc_i$ has unsupported conditions, the algorithm inserts new actions to support them (steps 6–7), generating new plans. If $acts\_to\_allocate_i$ is empty, step 10 selects the action `a` with the maximal allocation priority, which indicates the action to be allocated next (see below for an explanation of its calculus). Otherwise, `a` is extracted from $acts\_to\_allocate_i$ (step 12). If `a` is mutex in $Alloc_i$, it is removed or postponed depending on whether `a` is non-obligatory or obligatory, respectively (steps 13–17). If `a` is non-obligatory, it could be a bad choice in the relaxed plan, but if it is obligatory we know that it must be present in the plan, so it is not removed. Steps 18–23 try to allocate `a`. If `a` has unsupported conditions (it is not applicable), step 23 inserts new actions to support them, generating new plans.

The detection of mutex between actions in step 13 requires a deeper analysis. Although the spike construction has not calculated any mutex, they can still be deduced in the search. Graphplan defines two ways in which actions can be mutex: i) interference (one action has effects conflicting with conditions/effects of the other), and ii) competing needs (the conditions of the actions cannot hold simultaneously). On one hand, interference entails a static condition that depends on the definition of the actions. This information is static and is provided by TIM [18], which is currently been extended to deal with level 3 durative actions. On the other hand, the algorithm will never allocate actions with competing needs because their conditions cannot be present simultaneously in the same state. Therefore, the algorithm uses a *lazy* schema of mutex, which are only calculated between actions that are about to interact.

[2] For simplicity, the algorithms always consider the minimal cost best, assuming a minimisation problem. This does not reduce the generality of the algorithms, because a maximisation problem can be transformed into a minimisation one by multiplying all the costs with -1.

1: $set\_of\_plans \leftarrow \Pi$, generated in Algorithm 2
2: **while** $set\_of\_plans \neq \emptyset$ **do**
3: extract the lowest cost Π_i from set_of_plans
4: **if** $Alloc_i$ supports all the problem goals **then**
5: exit with success
6: **else if** $\forall \mathtt{a_j} \in Alloc_i \mid \exists \mathtt{p} \in \{SCond(\mathtt{a_j}) \cup Inv(\mathtt{a_j}) \cup ECond(\mathtt{a_j})\}$ that is not supported **then**
7: $\forall \mathtt{b_k}$ that supports p: insert Π_k into set_of_plans with $\mathtt{b_k}$ in $acts\_to\_allocate_k$
8: **else**
9: **if** $acts\_to\_allocate_i = \emptyset$ **then**
10: $\mathtt{a} \leftarrow \arg\max_{\forall \mathtt{a_j} \in Relax_i \text{ which can start at } time\_of\_execution_i}$(allocation priority)
11: **else**
12: extract a from $acts\_to\_allocate_i$
13: **if** a is mutex in $Alloc_i$ **then**
14: **if** a is non-obligatory in Π_i **then**
15: remove a from $Relax_i$
16: **else**
17: postpone start time of a in $Relax_i$
18: **else if** a is applicable **then**
19: allocate a in $Alloc_i$ at $time\_of\_execution_i$
20: **if** $acts\_to\_allocate_i = \emptyset$ **then**
21: regenerate $Relax_i$ from the current state
22: **else**
23: $\forall \mathtt{b_k}$ that makes a applicable: insert Π_k into set_of_plans with $\mathtt{b_k}$ in $acts\_to\_allocate_k$
24: update $time\_of\_execution_i$

Algorithm 3: Generation of a plan. Planning and allocating actions.

The algorithm has two branching points (steps 7 and 23), where it becomes necessary to insert new actions in new plans to support unsupported conditions. For each new action $\mathtt{a_j}$ supporting a condition, a new plan Π_j with action $\mathtt{a_j}$ marked as obligatory and inserted into $acts\_to\_allocate_j$ is generated. Moreover, the algorithm has two points of selection: steps 3 and 10. Step 3 selects the plan Π_i with the lowest cost from set_of_plans, where the cost has the traditional form $f(n) = g(n) + h(n)$:

$$cost(\Pi_i) = cost(Alloc_i \cup acts\_to\_allocate_i) + cost(Relax_i^{'})$$

$cost(Alloc_i \cup acts\_to\_allocate_i)$ is calculated by evaluating the numeric variables (after applying the actions present both in $Alloc_i$ and $acts\_to\_allocate_i$) in the optimisation function. The calculus of $cost(Relax_i^{'})$ requires two steps: i) the generation of a relaxed plan $Relax_i^{'}$ from the state achieved by the execution of $Alloc_i \cup acts\_to\_allocate_i$, and ii) the evaluation of the cost of the variables (after executing $Relax_i^{'}$) in the optimisation function. The main reason to generate $Relax_i^{'}$ is that it represents a better estimation than $Relax_i$ of a plan to achieve the problem goals and, consequently, of the remaining cost. $Relax_i$ was initially generated from the initial state, and this may be considerably different to the current state. In particular, the more the algorithm advances in time, the less precise $Relax_i$ becomes. Therefore, whenever the state changes (after allocating one action) and no actions remain in $acts\_to\_allocate_i$, the algorithm replaces $Relax_i$ with the regenerated plan $Relax_i^{'}$ (see steps 20–21). This new relaxed plan will better take advantage of the current state, improving the estimations and helping the plan generation.

Step 10 selects the action from $Relax_i$ with the maximal priority to be allocated at the current $time\_of\_execution_i$. Currently, this value prioritises the action that can be executed in the current state and minimises the number of mutex with the remaining actions[3].

5 Discussion

The approach described in this paper represents our first attempt to extend a temporal planner to deal with numeric variables in multiobjective planning, while solving some of the initial inefficiencies of TPSYS. Therefore, there are still some open points which require further investigation:

- Considering the numeric conditions of actions in the estimation calculus as in metric-FF. In the spike construction during the first stage, the numeric effects are considered without any relaxation, but not the conditions. This may make the estimations more optimistic than they really are. Additionally, these conditions could be also taken into account when generating the relaxed plans in the second stage to make them more precise. Particularly, the relaxation of the numeric conditions may lead to an empty relaxed plan in some cases. Let us assume one problem in which all the goals are numeric, i.e. `(fuel plane)` $\geq$ `500`, `(profit)` `>` `100`, etc. In this case, the relaxed plan contains no actions because the propositional goals are supported in an empty plan. Consequently, the generation of a plan starts from scratch, without an outline that helps build the plan.
- Defining a heuristic function to prune actions when supporting propositions. The algorithm for the generation of a plan inserts into the plan space as many new plans as actions support the unsatisfied propositions. This is essential in a complete approach, but the resulting branching factor can be prohibitive, specially when supporting numeric conditions that require one precise numeric value. Let us assume a (sub)goal like `(profit)` `=` `100`. In that case, we might apply many actions with $\{+=, -=, *=, /=\}$ effects on the variable `(profit)`. Although simple heuristics, like selecting $+=$ or $*=$ effects with maximum right hand side first as Hoffmann proposes in [4], can be used, they are not always enough. For instance, one alternative is first to apply $*=$ effects because the increment is bigger, and then tune the value with $+=$ or $-=$ effects. However, a second alternative can apply first $+=$ effects and then apply $*=$ effects to make the increment still bigger. Further, the order of application of these effects can vastly modify the length, complexity and cost of the plan.
- Improving the heuristic estimations in the two points of selection of the algorithm to generate a plan (see Algorithm 3). These two points extract the plan of minimal cost and the action with the maximal priority to be allocated. On one hand, improving the estimation of the plan cost requires a more precise analysis of the numeric conditions and effects, as indicated above in the first point. On the other hand, improving the estimation to select the next action requires to analyse both the features of the actions (local factors, such as cost, unsupported conditions, etc.) and the structure of the relaxed plan (global factors, such as causal links or mutex the action imposes in the plan).
- Dealing with continuous effects. Currently, the algorithm only guarantees the numeric conditions in the extreme points of the execution of the action, i.e. at the start and end points. However, a more complex model could consider additional constraints on the numeric variables all over the execution of the action. For instance, a variable `(fuel)` could be kept within an interval during the execution of an action for `fly`. Additionally, these conditions can represent complex functions such as linear, quadratic, etc.

[3] The original TPSYS estimation considers more local factors, such as the cost to make the action applicable, the number of direct successor actions (causal link dependencies) in $Relax_i$, the number of direct successors that meet, etc. We are investigating these and other factors to improve the estimation of this priority.

6 Conclusions and related work

The management of numeric variables and multiobjective optimisation functions is not yet a widely explored field in AI planning. However, in the last few years some attempts to extend the capabilities of the planners in that direction have been carried out. One of the first works to include reasoning under resource constraints in a Graphplan approach was done by Koehler in [19]. In her work, actions provide, produce or consume resources (expressed as numeric variables), but the assignment operators in the effects are restricted to $\{:=, + =, - =\}$. The domain of the variables is also represented by an interval through the construction of a classical planning graph. The search is performed in the (iterative deepening) Graphplan's way. Consequently, the plan is only *optimised* in terms of the number of planning steps, and no other optimisation criterion is considered. More recent works are based on heuristic planning. GRT-R [11], and its extension to deal with multiobjective planning MO-GRT [5], include information on resources to construct the heuristic that estimates the distance between each planning state and the goals. Similarly to our approach, each proposition has associated a cost-vector which is an estimate of the total cost of achieving that proposition. However, there is an important difference: different values for the variables are not encoded as an interval and one proposition can have different vectors that correspond to alternative ways of achieving the proposition, thus increasing the storage requirements. Sapa [12] and TP4 [13] basically use the numeric resource constraints as a measure to adjust the heuristic estimations. On one hand, the idea in Sapa is to preprocess the problem specification to find out the maximal increment in the resources levels. Next, this increment is used to readjust the estimations according to the resource consumptions. On the other hand, TP4 uses the resource constraints as a way to limit the set of actions that can be executed concurrently and avoid search. metric-FF [4, 14] follows the line of heuristic planning and applies the idea of relaxing the delete effects to numeric variables. Similarly to [19], the assignment operators in the effects are restricted to $\{+ =, - =\}$. One good property of metric-FF is that it considers the numeric conditions in the estimation, thus improving the *informedness* of the estimations. Although duration could be managed in principle as other numeric variables, the sequential approach of metric-FF makes this feature useless, which may degrade the quality of the plans.

This paper has presented an attempt to build a planner to deal with numeric variables, extending the work done in [16]. As a straight consequence of the numeric management, the planner can cope with problems with multiobjective optimisation criteria, thus giving the user more opportunities to optimise plans. The basic idea is to associate a vector of numeric variables to each proposition/action that indicate the estimated value for those variables in the achievement of that proposition/action. It is important to note that no special distinction is done now for the duration of actions, which is managed exactly as the rest of numeric variables. Therefore, in this approach the term of temporal planning is subsumed by the term of planning with numeric variables.

The process consists of dividing the planning algorithm into several stages. First, a stage constructs two spike vectors which encode the classical information stored in planning graphs, while estimating the optimistic and pessimistic values for the variables and the cost of the actions *w.r.t.* the problem optimisation criterion. Second, a stage performs the search process to generate a plan. This stage uses an initial relaxed plan, calculated in a backward way, as an outline of the final plan. This outline of the plan presents two important benefits: i) it prevents the planner from starting search from an empty plan, and ii) it provides useful information to determine the actions to be allocated next, and to estimate the distance from a given state to the goals. Unlike other approaches, this estimation does not relax any of the $\{:=, + =, - =, * =, / =\}$ effects, though the numeric conditions are not considered in the calculus of the relaxed plans. We are currently implementing and testing the algorithms, which are part of our ongoing work. In consequence, there exist some limitations that require additional investigation as discussed in section 5.

7 Acknowledgments

The work of the first author has been partially supported by the Spanish MCyT under projects DPI2001-2094-C03-03, TIC2001-4936-E and TIC2002-04146-C05-04, and by the Universidad Politécnica de Valencia under projects 20010017 and 20010980.

References

1. Fikes, R., Nilsson, N.: STRIPS: a new approach to the application of theorem proving to problem solving. Artificial Intelligence **2** (1971) 189–208
2. Pednault, E.: ADL: Exploring the middle ground between strips and the situation calculus. In: Proc. Int. Conference on Principles of Knowledge Representation and Reasoning (KR-89), San Francisco, CA, Morgan Kaufmann (1989) 324–332
3. McDermott, D.: PDDL - The Planning Domain Definition Language. AIPS-98 Planning Competition Committee. (1998)
4. Hoffmann, J.: The Metric-FF planning system: Translating "ignoring delete lists" to numeric state variables. to appear in Journal of Artificial Intelligence Research (2003)
5. Refanidis, I., Vlahavas, I.: Multiobjective heuristic state-space planning. to appear in Artificial Intelligence **145(1-2)** (2003) 1–32
6. Blum, A., Furst, M.: Fast planning through planning graph analysis. Artificial Intelligence **90** (1997) 281–300
7. Bonet, B., Geffner, H.: Planning as heuristic search. Artificial Intelligence **129** (2001) 5–33
8. Hoffmann, J., Nebel, B.: The FF planning system: Fast plan generation through heuristic search. Journal of Artificial Intelligence Research **14** (2001) 253–302
9. Smith, D., Weld, D.: Temporal planning with mutual exclusion reasoning. In: Proc. 16th Int. Joint Conference on AI (IJCAI-99), Stockholm, Sweden (1999) 326–337
10. Garrido, A., Fox, M., Long, D.: A temporal planning system for durative actions of PDDL2.1. In Harmelen, F.V., ed.: Proc. European Conference on AI (ECAI-2002), Amsterdam, IOS Press (2002) 586–590
11. Refanidis, I., Vlahavas, I.: Heuristic planning with resources. In Horn, W., ed.: Proc. 14th European Conference on AI (ECAI-2000), IOS Press (2000) 521–525
12. Do, M., Kambhampati, S.: Sapa: a domain-independent heuristic metric temporal planner. In Cesta, A., Borrajo, D., eds.: Proc. European Conference on Planning (ECP-2001). (2001) 109–120
13. Haslum, P., Geffner, H.: Heuristic planning with time and resources. In Cesta, A., Borrajo, D., eds.: Proc. European Conference on Planning (ECP-2001). (2001) 121–132
14. Hoffmann, J.: Extending FF to numerical state variables. In Harmelen, F.V., ed.: Proc. European Conference on AI (ECAI-2002), Amsterdam, IOS Press (2002) 571–575
15. Fox, M., Long, D.: PDDL2.1: an extension to PDDL for expressing temporal planning domains. Technical report, University of Durham, UK (2001)
16. Garrido, A., Onaindía, E.: On the application of least-commitment and heuristic search in temporal planning. In: Proc. Int. Joint Conference on AI (IJCAI-2003), San Francisco, CA, Morgan Kaufmann (2003) 942–947
17. Fox, M., Long, D.: Efficient implementation of the plan graph in STAN. Journal of AI Research **10** (1999) 87–115
18. Fox, M., Long, D.: The automatic inference of state invariants in TIM. Journal of AI Research **9** (1998) 367–421
19. Köehler, J.: Planning under resource constraints. In: Proc. European Conference on AI (ECAI-98). (1998) 489–493