

Planning with Numeric Variables in Multiobjective Planning

Antonio Garrido¹ and Derek Long²

Abstract. The purely propositional representation traditionally used to express AI planning problems is not adequate to express numeric variables when modeling real-world continuous resources. This paper presents a heuristic planning approach that uses a richer representation with capabilities for numeric variables, including durations of actions, and multiobjective optimisation. The approach consists of two stages. First, a spike construction process estimates the values of the variables associated with propositions/actions, without relaxing numeric effects in the calculus of the estimation. 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 and its successors. 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 [7, 10]. Although in a propositional representation planning graphs have become tremendously useful to estimate when propositions/actions are given [1, 7, 11], when dealing with time and numeric variables (i.e. fuel level, energy, profit, etc.), where the domain of the variables is usually continuous, the convenience of these graphs decreases. First, the role of the levels becomes unclear as they are not longer equidistant [5]. Second, a traditional planning graph cannot be directly used when actions impose complex constraints (inequalities) on the variables: i) conditions such as `(fuel plane) ≥ 100` or `(free-space truck) > 50`, and ii) effects such as `decrease (fuel plane) 10` or `scale-up (profit) 1.5`.

We present a heuristic planning approach with capabilities for numeric variables, including duration on actions, and multiobjective optimisation to manage level 3 durative actions of PDDL2.1 [4]. Level 3 of PDDL2.1 allows a combination of logical (propositional) and numeric features on actions, which have local conditions and effects. In addition, this work extends and stresses on the initial ideas for heuristic search in PDDL2.1 temporal planning of TPSYS [5], and attempts to solve its main inefficiencies and limitations.

2 A BRIEF REVIEW OF TPSYS

TPSYS is a Graphplan-based temporal planner that manages the non-conservative model of durative actions of PDDL2.1 and consists

of three stages [5]. The first stage is a preprocessing stage that calculates the action/action and proposition/action *static* mutex relations, which 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. A lot of effort is wasted again in the mutex calculus between actions which will not interact in the plan. A new indication of **inefficiency** arises because of the size of the planning graph, which may contain many temporal levels when the greatest common divisor of the durations is 1, thus increasing the complexity of the second stage.

The third stage performs the search of a temporal plan in two steps. 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. In addition, this search entails a **limitation**: plans are optimised only *w.r.t.* makespan. No other criteria, such as resource consumption or action cost can be used, thus limiting the capabilities of the planner.

3 NUMERIC VARIABLES IN ACTIONS

In this paper, we use durative actions as defined in PDDL2.1, with three types of conditions: *SCond(a)*, *Inv(a)* and *ECond(a)*, with the conditions to be guaranteed at the *start*, *over* the execution, and *end* of *a*, respectively. In a numeric approach, each condition can be either a classical propositional condition or a numeric condition that expresses a constraint as a tuple $\langle f\text{-exp1}, \text{binary-comp}, f\text{-exp2} \rangle$, where *f-exp1* and *f-exp2* represent functional expressions (which may contain numeric variables), and *binary-comp* $\in \{<, \leq, =, >, \geq\}$ is a binary comparator. Additionally, durative actions have two types of effects: *SEff(a)* and *EEff(a)*, with the effects to be asserted at the *start* and *end* of *a*, respectively. Each effect can be either a positive (*SAdd(a)*, *EAdd(a)*) or negative (*SDel(a)*, *EDel(a)*) propositional effect or a numeric effect as a tuple $\langle f\text{-head}, \text{assign-op}, f\text{-exp} \rangle$, 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 1 shows an example with three actions of the `zenotravel` domain with numeric conditions and effects used in

¹ Universidad Politecnica de Valencia (Spain), email: agarrido@dsic.upv.es

² University of Strathclyde (UK), email: derek.long@cis.strath.ac.uk

```

(: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 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))))
                (* (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 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)))

```

Figure 1. Actions with numeric capabilities used in IPC-2002.

IPC-2002³. For instance, action `fly` requires the plane to be in the origin (propositional condition), but also to have enough fuel to perform the flight (numeric condition). On the other hand, `fly` makes the plane to be in the destination (propositional effect), increases the variable `total-fuel-used` and decreases the level of fuel of the plane (numeric effects). By default, we include the variable `(total-time)`, modified by the field `:duration`, that represents the makespan of the plan. Note that this allows us to treat the duration as any other numeric variable. Different actions may present different numeric effects: fixed for all actions of an operator, different for each action of an operator (argument dependent), or even different for the same action (depending on the state where they are applied). In addition to this, each planning problem can define a multiobjective metric function to assess the quality of the plan, such as `(+ (* 4 (total-time)) (* 0.005 (total-fuel-used)))`. 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

Unlike TPSYS, the new approach does not generate a real temporal planning graph. This allows to overcome two of the most important inefficiencies of TPSYS: i) the calculus and propagation of mutex, and ii) the extension of a temporal planning graph [5]. Instead of extending a temporal graph, we generate one spike vector for propositions and one for actions. 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.

4.1 First stage. Spike construction

The first stage uses two spikes that encode the information about propositions/actions, thus reducing the storage requirements of a

³ More information about the domains of the IPC-2002 in: <http://planning.cis.strath.ac.uk/competition>

planning graph. Each proposition/action is associated with a vector of tuples $\langle f\text{-head}_i, \min_value_i, \max_value_i \rangle$, where $f\text{-head}_i$ is the i^{th} numeric variable and \min_value_i (\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: for all  $p \in \mathcal{I}_s$  do
2:    $prop\_state \leftarrow prop\_state \cup \{p\}$ 
3:   for all  $f\text{-head}_{p,i}$  do
4:     update  $f\text{-head}_{p,i}$  with value of  $f\text{-head}_i$  in  $\mathcal{I}_s$ 
5: while new actions can be applied from  $prop\_state$  do
6:   for all  $a \mid \{SCond(a) \cup Inv(a)\} \in prop\_state$  do
7:     insert  $a$  in the spike of actions
8:     for all  $f\text{-head}_{a,i}$  do
9:       update  $f\text{-head}_{a,i}$  with  $\cup^T$  value  $(f\text{-head}_{p_j,i})$ 
           $\forall p_j \in SCond(a) \cup Inv(a)$ 
10:    for all  $p \in \{SAdd(a) \cup EAdd(a)\}$  do
11:      insert  $p$  in the spike of propositions (if not present)
12:       $prop\_state \leftarrow prop\_state \cup \{p\}$ 
13:      for all  $f\text{-head}_{p,i}$  do
14:        update  $f\text{-head}_{p,i}$  with  $\cap^T$  effect  $(f\text{-head}_{a_j,i})$ 
           $\forall a_j$  supporting  $p$ 

```

Algorithm 1: Spike construction.

For each initial proposition p , the algorithm initialises the numeric variables $f\text{-head}_{p,i}$ 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 5-14 construct the spike structure until no new actions can be generated. For each action a that starts, steps 8-9 estimate 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, steps 13-14 estimate the minimal and maximal values of the variables for propositions as the *temporal intersection* (\cap^T) 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 such as Sapa, TP4 or metric-FF [2, 6, 7]. 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 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 like `board` or `fly`, but not `refuel`).

4.2 Second stage. Search of a plan

This stage is inspired by the two-step search of TPSYS. 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 function.

4.2.1 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 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:  $\Pi \leftarrow \{IS \cup FS\} \{obligatory\ actions\}$ 
2:  $queue\_acts \leftarrow FS$ 
3: while  $queue\_acts \neq \emptyset$  do
4:   extract  $a$  from  $queue\_acts$ 
5:   for all  $p \in \{SCond(a) \cup Inv(a) \cup ECond(a)\}$  do
6:     if  $p$  is not supported in  $\Pi$  then
7:        $b \leftarrow \arg \min(\text{cost of execution of } b_i)$ 
           $\forall b_i \text{ which supports } p$ 
8:       if  $b$  is the only action supporting  $p \wedge a$  is obligatory then
9:         mark  $b$  as obligatory in  $\Pi$ 
10:       $\Pi \leftarrow \Pi \cup \{b\} \{no\ commitment\ on\ execution\ time\}$ 
11:       $queue\_acts \leftarrow queue\_acts \cup \{b\}$ 

```

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 execution⁴. 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* [5] to indicate that such an action must be present in all the plans because it is the only way to support the problem goals⁵ (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 quite 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 in advance [2].

4.2.2 Generation of a plan. Planning and allocating actions

The plan is generated through a plan space search in a structure called set_of_plans that contains all the generated plans $\{\Pi_i\}$. Actions in each Π_i are divided into two disjunctive sets: $Relax_i$ and $Alloc_i$.

⁴ 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.

⁵ The idea of calculating actions that must be present in the plan has been previously explored in [9]. Although our calculus does not perform any real reasoning on landmarks, in many cases it provides very useful information with barely computational effort.

$Relax_i$ 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 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.

```

1:  $set\_of\_plans \leftarrow \Pi$ , generated in Algorithm 2
2: while  $set\_of\_plans \neq \emptyset$  do
3:   extract the lowest cost  $\Pi_i$  from  $set\_of\_plans$ 
4:   if  $Alloc_i$  supports all the problem goals then
5:     exit with success
6:   else if  $\forall a_j \in Alloc_i \mid \exists p \in \{SCond(a_j) \cup Inv(a_j) \cup ECond(a_j)\}$ 
     that is not supported then
7:      $\forall b_k$  that supports  $p$ : insert  $\Pi_k$  into  $set\_of\_plans$  with  $b_k$  in
      $acts\_to\_allocate_k$ 
8:   else
9:     if  $acts\_to\_allocate_i = \emptyset$  then
10:       $a \leftarrow \arg \max(\text{allocation priority})$ 
           $\forall a_j \in Relax_i \text{ which can start at } time\_of\_execution_i$ 
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  $\Pi_i$  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:              $\forall b_k$  that makes  $a$  applicable: insert  $\Pi_k$  into  $set\_of\_plans$  with
              $b_k$  in  $acts\_to\_allocate_k$ 
24:           update  $time\_of\_execution_i$ 

```

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

The idea is to simulate the real execution of Π_i , progressively taking care of the actions that 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 a more detailed explanation). Otherwise, a is extracted from $acts_to_allocate_i$ (step 12). If a is mutex⁶ 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, step 23 inserts new actions to support them, generating new plans.

The algorithm has two branching points (steps 7 and 23), where it inserts new actions in new plans to support conditions. For each new action a_j supporting a condition, a new plan Π_j with action a_j marked as obligatory and inserted into $acts_to_allocate_j$ is generated. a_j is inserted as obligatory to guarantee that in the plan the unsupported condition will be satisfied by a_j . Further, 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 is:

⁶ The mutex information is provided by TIM [3], which has been extended to deal with level 3 durative actions. Therefore, our approach uses a *lazy* schema of mutex, which are only calculated between actions that are about to interact.

$$\text{cost}(\Pi_i) = \text{cost}(\text{Alloc}_i \cup \text{acts_to_allocate}_i) + \text{cost}(\text{Relax}'_i)$$

$\text{cost}(\text{Alloc}_i \cup \text{acts_to_allocate}_i)$ is calculated by evaluating the numeric variables (after applying the actions present both in Alloc_i and $\text{acts_to_allocate}_i$) in the optimisation function. Analogously, $\text{cost}(\text{Relax}'_i)$ is calculated, where Relax'_i is formed by: i) the actions necessary to make $\text{acts_to_allocate}_i$ applicable from the state achieved by Alloc_i , and ii) the actions necessary to achieve the problem goals from the state achieved by $\text{acts_to_allocate}_i$. It is important to note that Relax'_i is a better estimation than Relax_i 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 $\text{acts_to_allocate}_i$, 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 $\text{time_of_execution}_i$. This value prioritises the action that can be executed in the current state (all its conditions are present) that minimises the number of mutex with the remaining actions in Relax_i .

5 EXPERIMENTAL RESULTS

In this section we have extended TPSYS with the new search approach. To date our implementation only provides plans with minimal action overlapping, which significantly reduces the quality of the plans. Figure 2 shows some comparisons (execution time and plan quality) for TPSYS, LPG-speed and MIPS in the `zenotravel` and `satellite` domains used in IPC-2002, which try to optimise both the makespan and total fuel used. The experiments for TPSYS were run on a Pentium IV 2 GHz with 512 Mb, whereas the results for LPG-speed and MIPS were provided in the competition (run in a slightly slower computer with 1 Gb).

TPSYS behaves quite well in `zenotravel` (Figure 2-ab), showing a scalable performance and better times than MIPS. Although the current plans of TPSYS are still highly sequential, it provides better plans than LPG.speed in most of the problems. In the `satellite` domain (Figure 2-cd), LPG.speed has better results, but TPSYS shows again a scalable performance. The times of TPSYS are faster than MIPS, but the plans of MIPS have better quality.

6 DISCUSSION

Our approach to deal with numeric variables in multiobjective planning has some open points that require further investigation:

- The spike construction considers the numeric effects of the actions without any relaxation, but no their numeric 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 a problem with the only goal $(\text{profit}) > 100$. In this case, the relaxed plan contains no actions because the propositional goals are supported in an empty plan. Thus, the generation of a plan starts from scratch, without an outline that helps build the plan.

- The branching factor due to actions with numeric effects might be prohibitive, specially when supporting conditions that require one precise numeric value. Let us assume a goal like $(\text{profit}) = 100$. We might apply actions with $\{+ =, - =, * =, / =\}$ effects on the variable (profit) . Although simple heuristics, like selecting $+ =$ or $* =$ effects with maximum right hand side first as proposed in [7], can be used, they are not always enough. For instance, one alternative is first to apply $* =$ effects, and then tune the value with $+ =$ or $- =$ effects. However, a second alternative might apply first $+ =$ effects and then apply $* =$ effects. Further, the order of application of these effects can modify the length, complexity and cost of the plan.
- The heuristic selection of actions may get stuck in the wrong path while generating the plan, which entails to visit the same states within a loop. Although memoization techniques [1] are helpful to avoid this situation, they get more complex when dealing with numeric variables: for identical propositional states the values of the numeric variables can be different. This involves a huge explosion in the space requirements and new techniques must be explored.
- Currently, the algorithm does not deal with continuous effects and only guarantees the numeric conditions in the extreme points of the execution of the action, i.e. at the start and end. A more complex model should consider additional constraints on the variables all over the execution of the action (for instance, when the value for `fuel` must be in an interval during the execution of `fly`).

7 CONCLUSIONS AND RELATED WORK

In the last few years, some attempts to extend the capabilities of the planners to manage numeric variables and multiobjective optimisation functions have been carried out in AI planning. One of the first works to include reasoning under resource constraints in a `Graphplan` approach was done in [8]. In this 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, but the plan is only optimised in terms of the number of planning steps. More recent works are based on heuristic planning. GRT-R and its extension to deal with multiobjective planning MO-GRT [10], include information on resource consumption 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 [2] and TP4 [6] use the constraints on resource consumption to adjust the heuristic value that estimates the length of the plan. On one hand, Sapa preprocesses 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 actions that can be executed concurrently and avoid search. Unlike these planners, metric-FF [7] also considers the numeric conditions of the actions, but it still ignores the decreasing effects in the heuristic estimation. Similarly to [8], the assignment operators in the effects are restricted to $\{+ =, - =\}$. One interesting property of metric-FF is that it considers the numeric conditions in the estimation, thus improving the *informedness* of the estimations. Although duration

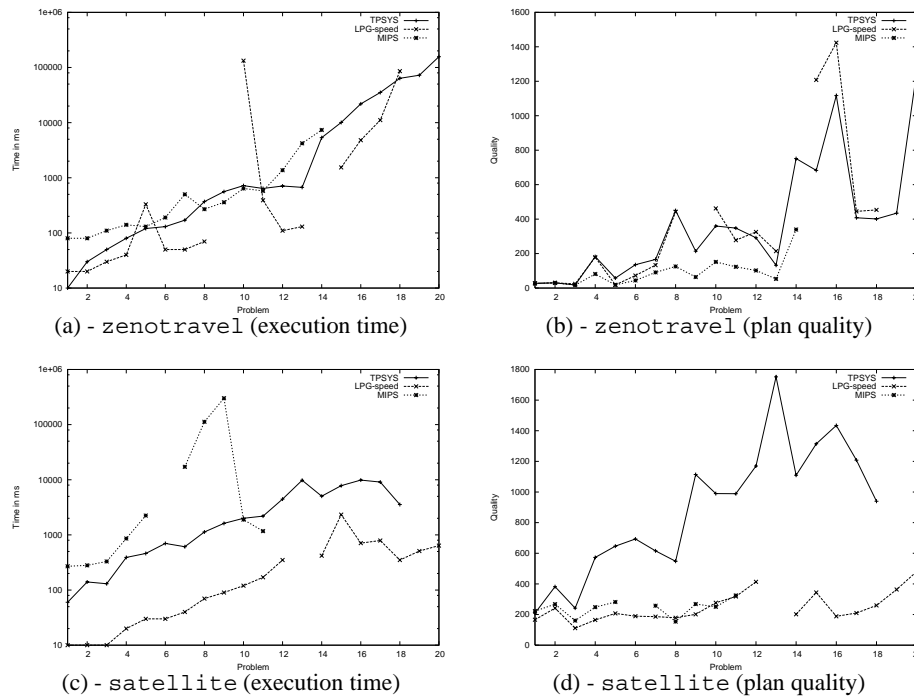


Figure 2. Comparison of the new approach for search implemented in TPSYS vs. LPG.speed and MIPS.

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 proposed a planning approach to deal with numeric variables. As a consequence of the numeric management, the planner can cope with problems with multiobjective optimisation criteria, thus giving the user more opportunities to optimise plans. Briefly, the main contributions of the paper have been the description of:

- The basic idea 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.
- A first stage that 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.
- A second stage that 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 working on several parts of the implementation

of the algorithm to find plans with more actions in parallel. This will lead to significant improvements in the quality of the plans. As discussed in section 6, there exist some limitations that require additional investigation, which is part of our future work.

8 ACKNOWLEDGMENTS

The work of the first author has been partially supported by the Spanish MCyT under projects DPI2001-2094-C03-03 and TIC2002-04146-C05-04, and by the project UPV-20020681.

REFERENCES

- [1] A.L. Blum and M.L. Furst, ‘Fast planning through planning graph analysis’, *Artificial Intelligence*, **90**, 281–300, (1997).
- [2] M.B. Do and S. Kambhampati, ‘Sapa: a domain-independent heuristic metric temporal planner’, in *ECP-2001*, pp. 109–120, (2001).
- [3] M. Fox and D. Long, ‘The automatic inference of state invariants in TIM’, *JAIR*, **9**, 367–421, (1998).
- [4] M. Fox and D. Long, ‘PDDL2.1: an extension to PDDL for expressing temporal planning domains’, Tech. report, Univ. Durham, UK (2001).
- [5] A. Garrido and E. Onaindía, ‘On the application of least-commitment and heuristic search in temporal planning’, in *IJCAI-2003*, (2003).
- [6] P. Haslum and H. Geffner, ‘Heuristic planning with time and resources’, in *ECP-2001*, pp. 121–132, (2001).
- [7] J. Hoffmann, ‘The Metric-FF planning system: Translating “ignoring delete lists” to numeric state variables’, *JAIR*, **20**, 291–341, (2003).
- [8] J. Köehler, ‘Planning under resource constraints’, in *ECAI-98*, pp. 489–493, (1998).
- [9] J. Porteous, L. Sebastia, and J. Hoffmann, ‘On the extraction, ordering, and usage of landmarks in planning’, in *ECP-2001*, pp. 37–48, (2001).
- [10] I. Refanidis and I. Vlahavas, ‘Multiobjective heuristic state-space planning’, *Artificial Intelligence*, **145**(1-2), 1–32, (2003).
- [11] D.E. Smith and D.S. Weld, ‘Temporal planning with mutual exclusion reasoning’, in *IJCAI-99*, pp. 326–337, (1999).