On the Separation of Propositional and Numeric Planning in Realistic Problems

Antonio Garrido and Eva Onaindía

Dpto. Sistemas Informáticos y Computación
Universidad Politécnica de Valencia
Camino de Vera s/n, 46022 Valencia, Spain
Fax: (+34) - 963877359
{agarridot,onaindia}@dsic.upv.es

Abstract. State-of-the-art planners need to use different strategies to cope with the increasing complexity of realistic planning problems. These strategies tend to simplify the problem to make it more affordable. The separation of the planning and scheduling parts, the relaxation of the original problem, the application of heuristics, etc. are usual in modern planning. In this paper, we present an ongoing approach that divides the planning process into two consecutive stages. First, a sequential plan is generated by discarding the numeric conditions/effects of actions, thus ignoring the scheduling part of the problem. Second, this plan is parallelised while the satisfaction of numeric constraints is guaranteed, thus taking the scheduling part of the problem into account. Both stages use heuristic estimations calculated through a relaxed version of a planning graph that help reduce the search space.

1 Introduction

AI planning research has traditionally aimed at solving realistic problems. In this research line, last decade might be considered as the beginning of a new era for AI planning. Many of the traditional simplifications imposed to cope with realistic problems that have remained unaltered through many years have started to disappear. Planning graphs, satisfiability, local search, constraint satisfaction, model checking and heuristic techniques have been widely used to significantly improve the performance of the planning algorithms, thus allowing them to deal with more realistic capabilities, such as a richer action model that considers time, numeric variables, resources and multi-optimisation criteria [1–11]. In fact, currently the development of appropriate benchmarks for planning requires a considerable effort to reflect real application scenarios, such as domains for logistic problems, ground traffic on airports, space applications and control of satellites, etc. (see [12] and the IPC–04 domains[1] for more application examples). However, as the problems become harder and more complex, the difficulties to solve them become harder as well, and the algorithms need to come up with new strategies and techniques to enhance their performance. Modern planners, like most that participated in last IPC–04, use techniques that tend to simplify the original problem to reduce its complexity in different ways:

1. Separation of the planning and scheduling parts of the original problem, or at least, separation of the planning aspects from the domain modeling activities. The goal is to devote the planning effort in the efficient construction of plans and to use a specific scheduler to check whether the plan is schedulable or not [13, 14].
2. Relaxation of the problem constraints and use of heuristics. Generally, this technique builds a relaxed version of a planning graph that does not consider negative effects nor mutex relations and uses

[1] More information about the domains of the last International Planning Competition 2004 in http://ipc.icaps-conference.org

it to extract heuristic estimations based on distance to the goals, action cost, numeric values, etc. Moreover, the actions of the relaxed plan can also be used as a part of the solution plan. Hence, the main goal is to guide the heuristic search to the goals, avoiding the exploration of irrelevant actions/states [1, 15–23].
3. Utilisation of preprocessing techniques to speed up the planning process. The goal is to extract as much information as possible from the domain or problem structure: causal graphs, heuristic estimations, possible orderings, mutex relations, relevant actions, etc. [6–8, 10, 11, 13, 22, 23].
4. Detection and automatic learning of macro-actions. The goal is to reduce the planning effort by identifying and exploiting abstraction, action grouping and sequences (*plan prefixes*), thus helping reduce the search space (states change in a faster way and many intermediate states are *skipped*) [15, 24, 25].
5. Decomposition or reduction of the original problem. The goal is to create a reduced instance of the problem, solve it and then use the knowledge on the solution to assist when solving the original problem [16]. Similarly, partitioning the original problem into subproblems and solving them independently also shows effective: the detection of orderings among subgoals/actions, landmarks and global constraints is useful for more efficient planning [17–19, 26].

This paper presents an ongoing approach based on the work proposed in [27] that uses some of the first three previous techniques to deal with numeric variables (including time) in multiobjective planning. **First**, it disjoins the planning part from the scheduling one. The underlying idea is to separate the structural (propositional) part of the plan from the part dedicated to resource usage (numeric variables and resource management). **Second**, the process to find a plan has also been divided into two stages: i) generation of a sequential plan without considering numeric conditions/effects; and ii) parallelisation of this plan, including new actions to satisfy the numeric (resource) constraints. In both cases, heuristic estimations calculated through a relaxed version of a planning graph are used. These estimations allow to guide the search, whose search space is also reduced due to the fact that a process that calculates the irrelevant actions removes actions that are not necessary in the plan. **Third**, no mutex calculus is performed while planning. The mutex information is directly provided by TIM [28], a fully automatic system that extracts information about the structure of the domain and problem, as a preprocessing stage. Therefore, the algorithm uses a *lazy* schema of mutex, which are only calculated between actions that are about to interact. Consequently, no effort is wasted in the mutex calculus between actions which will not interact in the plan. This approach has some nice advantages:

- A sequential plan is calculated more easily than a parallel one. Clearly, no overlapping nor mutex relations have to be taken into consideration when actions are serially planned. Although there is no guarantee about the optimality of such a plan, its quality can be improved by means of an anytime behaviour.
- Some unsolvable situations can be detected more quickly. Those cases where the plan has no solution because propositional goals are unfeasible can be detected before starting to explore other (numeric) actions that deal with resources and have no influence in the problem solvability.
- After finding a feasible sequential plan, a lower bound can be easily derived to help the parallelisation process. Obviously, the precision degree of this bound depends on the problem metric to be optimised. For instance, in cases where the problem metric only represents the makespan, this bound will not be very informative (the makespan of the sequential plan is usually much longer than the makespan of the parallel plan). However, in cases where the problem metric is more action-dependent, this bound will be very precise and informative.

This paper is structured as follows. Section 2 briefly introduces the notation for expressing numeric variables (conditions and effects) in durative actions of PDDL2.x. The two stages of this approach are presented in sections 3 and 4; section 3 indicates the way the sequential plan is generated, whereas the algorithms to parallelise this plan are detailed in section 4. Section 5 reviews some related work and

discusses some points about separating out the planning and scheduling (numeric) parts. Finally, section 6 provides the contributions of the paper and shows the future work.

2 Numeric variables in actions

In this paper, we use durative actions as defined in PDDL2.1 and PDDL2.2 [4, 5], with three types of conditions: $SCond(\texttt{a})$, $Inv(\texttt{a})$ and $ECond(\texttt{a})$, 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 ⟨`f-exp1,binary-comp,f-exp2`⟩, 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: $SEff(\texttt{a})$ and $EEff(\texttt{a})$, with the effects to be asserted at the *start* and *end* of `a`, respectively. Each effect can be either a positive $(SAdd(\texttt{a}), EAdd(\texttt{a}))$ or negative $(SDel(\texttt{a}), EDel(\texttt{a}))$ propositional effect or a numeric effect as a tuple ⟨`f-head,assign-op,f-exp`⟩, 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`.

```
(: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 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. 1. Two typical actions of a logistics domain.

Fig. 1 shows an example of two actions (`fly` and `refuel`) of a logistics domain. `fly` requires the aircraft to be at the origin (propositional condition), but also to have enough fuel to perform the flight (numeric condition). On the other hand, `fly` makes the aircraft to be at the destination (propositional effect), increases the variable `total-fuel-used` and decreases the `level` of fuel of the aircraft (numeric effects). Similarly, `refuel` has both propositional and numeric conditions, but only numeric effects. A new variable `total-time`, which represents the makespan of the plan, is always included by default and modified by the field `:duration` present in each action. Note that this allows us to treat the duration as any other numeric variable with no special distinction. Moreover, 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.

The way the numeric variables are managed in this approach consists of associating each proposition/action with a vector of tuples $\langle$`f-head`$_i$`,min_value`$_i$`,max_value`$_i\rangle$, where `f-head`$_i$ is the ith numeric variable and `min_value`$_i$ (`max_value`$_i$) stands for the minimal (maximal) value estimated for that variable to achieve the proposition/action (see [27] for more details).

3 Generation of a sequential plan

The goal of this stage is to generate a sequential plan that works with causal constraints but without considering the numeric conditions and effects of actions. This plan is generated in a forward way through a plan space search in a structure called *set_of_plans* that contains all the generated plans $\{\Pi_i\}$. Each Π_i has two structures: Seq_i that contains the actions that form the sequential plan and determine the current state; and $Relax_i$ that contains a relaxed plan from the current state to the problem goals. A relaxed plan consists of a partially ordered set of actions, with no mutex considerations, connected by causal links in which all the propositional (sub)goals hold.

1: *set_of_plans* $\leftarrow$ empty plan with both Seq_i and $Relax_i$ empty
2: **while** *set_of_plans* $\neq \emptyset$ **do**
3: extract the lowest cost Π_i from *set_of_plans*
4: **if** Π_i is a solution plan **then**
5: exit with success
6: **else**
7: $Relax_i \leftarrow Relax(\Pi_i)$
8: $Cost_i \leftarrow Cost(\Pi_i)$
9: $RA_i \leftarrow Relevant\_Actions(\Pi_i)$ {*relevance analysis*}
10: $Generate\_Successor\_Plans(\Pi_i, RA_i)$

Algorithm 1: Generation of a sequential plan in a forward way.

The Algorithm 1 starts from an empty plan (step 1). Each iteration extracts the lowest cost Π_i from *set_of_plans* (step 3). If Π_i is a solution plan, the algorithm outputs with success (steps 4–5). Otherwise, the relaxed plan $Relax_i$ of Π_i is generated and its cost is calculated (steps 7–8). On one hand, the relaxed plan is generated as usual in planning literature [7, 27], using the actions that require less actions to be executed and have the best cost *w.r.t.* the problem metric. On the other hand, the cost of a plan is heuristically calculated as: $Cost(\Pi_i) = |Seq_i| + \omega|Relax_i|$, where $|Seq_i|$ and $|Relax_i|$ represent the number of actions in each structure, respectively, and ω allows to set the heuristic estimation[2]. Step 9 performs a relevance analysis by means of a backward process that only selects the relevant actions RA_i to achieve the problem goals from the current state. This helps reduce the branching factor when generating the successor plans of Π_i in step 10 (see Algorithm 2).

Algorithm 2 explores all the actions $\mathtt{a}_j$ in the relevant actions RA_i, performing the real search. If $\mathtt{a}_j$ is directly executable in Seq_i, a new plan Π_j is constructed with $\mathtt{a}_j$ (steps 1–3). Unfortunately, Seq_j could lead to a state already visited by previous plans, making the algorithm fall in an infinite loop. In order to avoid this, the algorithm needs to check whether the state of Seq_j is already memoized, discarding Π_j in such a case (steps 4–5). Otherwise, both $Relax(\Pi_j)$ and $Cost(\Pi_j)$ are calculated in steps 7–8. The algorithm uses a Hill-climbing variation. The strategy is to record into *set_of_plans* every successor

[2] Greater values of ω may find a solution faster but degrading its quality (in the current implementation $\omega = 3$).

1: **for all** $\mathtt{a}_\mathrm{j} \in RA_i$ **do**
2: **if** $\mathtt{a}_\mathrm{j}$ is executable in the current state of Seq_i **then**
3: construct a new Π_j from Π_i, with $Seq_j \leftarrow Seq_i \cup \{\mathtt{a}_\mathrm{j}\}$
4: **if** current state of Seq_j is already memoized **then**
5: discard Π_j
6: **else**
7: $Relax_j \leftarrow Relax(\Pi_\mathrm{j})$
8: $Cost_j \leftarrow Cost(\Pi_\mathrm{j})$
9: **if** $(Cost_j \leq Cost_i) \vee (Cost_j = \min_{\forall \Pi_\mathrm{k}}(Cost(\Pi_\mathrm{k})))$ **then**
10: $set\_of\_plans \leftarrow set\_of\_plans \cup \{\Pi_\mathrm{j}\}$
11: memoize current state of Seq_j

Algorithm 2: $Generate\_Successor\_Plans(\Pi_\mathrm{i}, RA_i)$.

$\Pi_\mathrm{j} \mid Cost(\Pi_\mathrm{j}) \leq Cost(\Pi_\mathrm{i})$, or the successor with minimal cost if all the successors have worse cost than $Cost(\Pi_\mathrm{i})$ (steps 9–10). Finally, step 11 memoizes the current state of Seq_j to avoid future loops.

The output of this stage is either a sequential plan that is *propositionally sound*, or failure. Intuitively in the former, the plan would be directly executable in a scenario without numeric variables, conditions and effects, such as a pure-STRIPS planning problem. Unfortunately, since Algorithm 2 is not complete (clearly, steps 9–10 do not record all the successor plans) there is no completeness guarantee. There exists, however, an important advantage here. If the problem is propositionally unsolvable, i.e. there are some propositional (sub)goals that cannot be simultaneously satisfied, this stage outputs that no feasible plan can be found. It must be noticed that this is stated before analysing numeric actions, which make the search space larger and have a negative impact in the algorithm performance, but have no influence in this kind of problem solvability.

Furthermore, this sequential plan provides the planner with an initial lower bound that can heuristically help the plan parallelisation stage. The quality of this bound highly depends on the problem metric and it can be more or less informative according to the criteria used. Particularly, problems where makespan plays an important role in the metric lead to bad informative bounds: makespan of a sequential plan is not representative for the same plan after being parallelised. On the contrary, problems where metric focuses on more action-dependent criteria, such as execution cost, fuel consumption, data stored, etc., provide good indicators on the quality of the plan, both in its sequential and parallel versions.

4 Plan parallelisation and satisfaction of numeric constraints

The goal of this stage is to put the actions of the sequential plan in parallel, including the necessary actions to satisfy the numeric constraints. Although there exist some algorithms to parallelise plans and to reorder actions within plans [29–31], the approach presented here cannot use them in a straightforward way. The critical difference relies on the initial plan to be parallelised; traditional research focuses on generating a parallel plan using as a basis an **executable** partial or total plan. This means that all the (sub)goals hold, and only a better ordering among actions is required. On the contrary, the initial plan used in this approach could have unsolved conditions that must be solved before the plan can become executable. Consequently, this stage also needs to include a search procedure for these situations.

As in the previous stage, a structure set_of_plans is required to store the plans $\{\Pi_\mathrm{i}\}$. Now, each Π_i is extended with a new structure called $Alloc_i$, which contains the actions that have been allocated in time and determine the current state. It is important to note that once an action is inserted into $Alloc_i$, it will never be removed. For simplicity, let us assume that actions in Seq_i are in the form $\langle \mathtt{a}_\mathtt{i}^1, \mathtt{a}_\mathtt{i}^2, \mathtt{a}_\mathtt{i}^3 \ldots \mathtt{a}_\mathtt{i}^\mathtt{n} \rangle$.

1: $set\_of\_plans \leftarrow \Pi_i$, generated by Algorithm 1 {$Alloc_i$ *is initially empty*}
2: **while** $set\_of\_plans \neq \emptyset$ **do**
3: extract the lowest cost Π_i from set_of_plans
4: **if** Π_i is a solution plan **then**
5: exit with success
6: **else**
7: **for all** $\mathtt{a}_i^j \in Seq_i$ not allocated yet, $j=1..|Seq_i|$ **do**
8: $T_{\mathtt{a}_i^j} \leftarrow Find\_Execution\_Time(\mathtt{a}_i^j, Seq_i, Alloc_i)$
9: **if** $\mathtt{a}_i^j$ is executable in the current state of $Alloc_i$ at time $T_{\mathtt{a}_i^j}$ **then**
10: $Alloc_i \leftarrow Alloc_i \cup \{\mathtt{a}_i^j\}$ {$\mathtt{a}_i^j$ *is inserted at time* $T_{\mathtt{a}_i^j}$}
11: mark $\mathtt{a}_i^j$ as allocated
12: **else**
13: **for all** numeric action $\mathtt{b}_k$ that makes $\mathtt{a}_i^j$ executable **do**
14: construct a new Π_k from Π_i, with $Seq_k \leftarrow \{\mathtt{b}_k\} \cup Seq_i$
15: $set\_of\_plans \leftarrow set\_of\_plans \cup \{\Pi_k\}$
16: go to step 2

Algorithm 3: Plan parallelisation and satisfaction of numeric constraints.

Algorithm 3 starts with the sequential plan generated in Algorithm 1 (step 1). Each iteration extracts the lowest cost[3] Π_i and checks if it is a solution plan (steps 3–5). If Π_i is not a plan solution, step 7 tries to allocate the actions in Seq_i. Step 8 (see Algorithm 4) finds the earliest time $T_{\mathtt{a}_i^j}$ when action $\mathtt{a}_i^j$ could start in $Alloc_i$. If $\mathtt{a}_i^j$ is executable in $Alloc_i$ at time $T_{\mathtt{a}_i^j}$, it is inserted into $Alloc_i$ and marked as allocated (steps 9–11). If $\mathtt{a}_i^j$ is not executable (because its numeric conditions are not satisfied), steps 13–15 insert new numeric actions $\mathtt{b}_k$ that make it executable, constructing new plans and inserting them into set_of_plans. Next, step 16 goes back to step 2 to resume the process with another plan.

1: **if** $Alloc_i = \emptyset$ **then**
2: return 0 {$\mathtt{a}_i^j$ *is the first action to be allocated*}
3: **else**
4: $T \leftarrow$ end time of the latest action $\mathtt{a}_i^k$, $k=1..j-1$, that has a causal dependency $\mathtt{a}_i^k \longrightarrow \mathtt{a}_i^j$
5: **if** $\mathtt{a}_i^j$ is not mutex at time T with any other action present in $Alloc_i$ **then**
6: return T
7: **else**
8: return earliest time $T' > T$ when $\mathtt{a}_i^j$ is not mutex with any other action present in $Alloc_i$

Algorithm 4: $Find\_Execution\_Time(\mathtt{a}_i^j, Seq_i, Alloc_i)$.

Intuitively, Algorithm 4 returns the earliest start time when action $\mathtt{a}_i^j$ could start. This time is 0 when $Alloc_i$ is empty (steps 1–2). If there exist allocated actions $\mathtt{a}_i^k$ that achieve conditions of $\mathtt{a}_i^j$ (causal dependency), the return value is the end time T of the latest $\mathtt{a}_i^k$ if no mutex[4] is present in $Alloc_i$ at that time (steps 4–6). The most complex case occurs when $\mathtt{a}_i^j$ is mutex in $Alloc_i$ at time T. In this case, the return value is the earliest $T' > T$ when $\mathtt{a}_i^j$ is not mutex in $Alloc_i$ (step 8).

[3] Unlike Algorithms 1–2, $Cost(\Pi_i)$ is now calculated by evaluating the actions in $Alloc_i$ and $Relax_i$ in the problem metric.

[4] Mutex information is calculated in a preprocessing stage by TIM [28]. Therefore, it is not necessary to perform any mutex calculus or propagation while planning actions.

This stage returns a parallel plan that may contain numeric actions, or failure. The previous algorithms cannot guarantee the completeness and optimality properties. Furthermore, the sequential plan might not be properly parallelised if no numeric actions are available to satisfy the numeric conditions. For instance, let us suppose that the available level of fuel for an aircraft is limited (no `refuel` action is available or it must be planned in a very distant city). If the aircraft runs out of fuel in a journey, the algorithm will not be able to find an executable plan to move the aircraft to the city to refuel and eventually, it will return failure.

5 Related work and discussion

Traditionally, researchers have found many difficulties to cope with realistic problems: how to model and define all the features of the problems, how to use the problem definition directly in the algorithms, how to improve the performance of these algorithms with planning techniques, how to integrate ad-hoc intelligent knowledge into these techniques, etc. Fortunately, last advances in planning algorithms have allowed the development of modern planners that overcome many of these difficulties. Currently, most state-of-the-art planners use heuristic techniques to guide the search process when finding the plan. The way they extract the heuristic information is quite similar in all of them: they usually build a planning graph without considering negative effects nor mutex relations [1, 2, 7, 8]. However, the way they apply this heuristic information differs from ones to others [3, 6, 7, 9].

Some researchers have tried to solve the planning problems *as are*, i.e. with both the planning and scheduling components. For instance, GRT-R and MO-GRT [9, 10] do not make any separation in the problem, but they enrich the heuristic estimations with information on resource consumption. metric-FF [8] follows the same heuristic line, relaxing the delete effects to numeric variables but considering the numeric conditions in the estimations to improve their *informedness*. However, temporal problems and durative actions do not fit well in the sequential approach of metric-FF. LPG [6] is based on stochastic local search techniques to move throughout the search space, but again no special distinction is done between planning and scheduling (the heuristic estimations also include the numeric information). YAHSP [18] extends the heuristics presented in FF [7], computing and using lookahead states and plans to improve the search strategy, but it does not solve temporal problems.

On the other hand, nearly the whole number of the planners that participated in the last IPC–2004 tend to simplify the original problem to make it easier and, consequently, more affordable. One alternative widely used is to identify and explore any kind of abstraction (macro-actions, meta-action sequences, state grouping, etc.), as used in Macro-FF and FAP [15, 24]. Planning decomposition is also useful to create reduced versions of the original problem, or subproblems which are solved independently, that will assist with solving the original problem (Marvin, SGPlan and Fast (Diagonally) Downward [16, 17, 22]). Other alternatives are more similar to the work presented in this paper: i) separating out the planning and scheduling parts of a planning problem (CRIKEY [13]), and ii) finding a sequential version of a plan that is later parallelised (P-MEP [23]). The advantages of doing this are twofold. First, the original search is divided into two stages. This shows effective because each stage involves a smaller search space than that of the original problem, as marked in [17]. Particularly, a sequential search proceeds in a more efficient way than a parallel one. Next, the process of parallelising a sequential plan is easier than generating a parallel plan from scratch. Second, different heuristic techniques can be used in each stage: i) planning-oriented heuristics, to find a sequential plan in a fast way trying to optimise the length of the plan; and ii) quality-oriented heuristics, to find a good schedulable parallel plan taking into consideration the problem metric defined by the user. However, this is not a drawback-free approach. Commonly, these two types of heuristics are conflicting; shorter plans are more expensive in terms of the problem metric. And it might be even worse. A good sequential plan may be impossible to be parallelised, and without feedback from one stage to the other, the problem turns unsolvable. Consequently, no completeness nor optimality properties can be guaranteed.

An additional drawback comes directly from the way the domain is defined. It is relatively easy to automatically discard the numeric variables in action conditions/effects when finding a sequential plan, and include them again when parallelising it. However, the domain modeller may choose to replace these variables with propositional predicates. For instance, function `(fuel ?aircraft)`$\in \mathbb{R}$ may be replaced with predicate `(fuel ?aircraft ?level)`, where `?level` has the propositional domain {`level1,level2,...leveln`}. In such a case, it is not easy nor intuitive to detect the actions that deal with resources and the separation into two stages has no beneficial effect.

6 Conclusions and future work

Typically, there are two key factors that determine the fulfillment of a plan: i) the actions that form the plan, and ii) the execution times of these actions. This paper has presented an extension of the approach proposed in [27] as part of an ongoing work that attempts to split a planning problem into **which** actions are going to be used and **when**. The main idea is to separate the planning part (ordering and causal constraints) from the scheduling part (numeric variables that usually involve resources) to solve realistic planning problems. As discussed in section 5, this separation depends on the domain definition, but if possible can help improve the search process. Hence, the main contributions of this paper have been the description of:

- The basic idea to separate the planning and scheduling (management of numeric variables) parts in a planning problem. This separation is trivial because it just consists of ignoring the numeric conditions and effects.
- A first stage that generates a sequential plan in a forward way. This stage focuses on the structural part of the plan, dealing with causal constraints and dependencies to solve the (sub)goals. The idea is to focus the planning effort on finding a more general plan, postponing the reasoning on numeric variables and actions. The algorithm uses a heuristic based on the length of the sequential+relaxed plans in a Hill-climbing variation. Additionally, it performs a simple relevance analysis in a backward way that helps reduce the branching factor.
- A second stage that parallelises the sequential plan. While it tries to allocate the actions as early as possible to compact the plan, it includes the necessary numeric actions to satisfy the numeric conditions. Now, the heuristic focuses on the problem metric instead of the length of the plan. The main disadvantage here is that a sequential plan might not be parallelised because of the lack of numeric actions to satisfy the numeric conditions.

This work has not been finished yet (some implementation tasks, mainly in the parallelisation stage, are still left) and some ideas on the definition and use of heuristics are not mature enough. Therefore, we cannot provide experimental results in the paper yet, which obviously are part of the current and future work. We are also interested in the development and application of new heuristic techniques in both stages, which can lead to some improvements in the quality of the final plan, and how more complex temporal constraints should be included in this approach.

Acknowledgments

This work has been partially supported by the Spanish government CICYT projects TIC2002-04146-C05-04 and DPI2001-2094-C03-03, by the Valencian government project GV04A-388 and by the Universidad Politécnica de Valencia under project 20020681.

References

1. Blum, A., Furst, M.: Fast planning through planning graph analysis. Artificial Intelligence **90** (1997) 281–300
2. Bonet, B., Geffner, H.: Planning as heuristic search: New results. In Biundo, S., Fox, M., eds.: Proc. European Conference on Planning (ECP-99), Springer (1999) 360–372
3. Edelkamp, S., Helmert, M.: The model checking integrated planning system MIPS. AI Magazine (2001) 67–71
4. Edelkamp, S., Hoffmann, J.: PDDL2.2: the language for the classical part of IPC–4. In: Proc. Int. Conf. on Automated Planning and Scheduling (ICAPS-2004) – International Planning Competition. (2004) 2–6
5. Fox, M., Long, D.: PDDL2.1: an extension to PDDL for expressing temporal planning domains. Journal of Artificial Intelligence Research **20** (2003) 61–124
6. Gerevini, A., Saetti, A., Serina, I.: Planning through stochastic local search and temporal action graphs in LPG. Journal of Artificial Intelligence Research **20** (2003) 239–290
7. Hoffmann, J., Nebel, B.: The FF planning system: Fast plan generation through heuristic search. Journal of Artificial Intelligence Research **14** (2001) 253–302
8. Hoffmann, J.: The Metric-FF planning system: Translating "ignoring delete lists" to numeric state variables. Journal of Artificial Intelligence Research **20** (2003) 291–341
9. 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
10. Refanidis, I., Vlahavas, I.: Multiobjective heuristic state-space planning. Artificial Intelligence **145(1-2)** (2003) 1–32
11. 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
12. Hoffmann, J., Edelkamp, S., Englert, R., Liporace, F., Thiebaux, S., Trug, S.: Towards realistic benchmarks for planning: the domains used in the classical part of IPC–4. In: Proc. Int. Conf. on Automated Planning and Scheduling (ICAPS-2004) – International Planning Competition. (2004) 7–14
13. Halsey, K.: The workings of CRIKEY – a temporal metric planner. In: Proc. Int. Conf. on Automated Planning and Scheduling (ICAPS-2004) – International Planning Competition. (2004) 35–37
14. Parker, E.: Combining backward-chaining with forward-chaining AI search. In: Proc. Int. Conf. on Automated Planning and Scheduling (ICAPS-2004) – International Planning Competition. (2004) 51–52
15. Camilleri, G., Zalaket, J.: FAP: Forward anticipating planner. In: Proc. Int. Conf. on Automated Planning and Scheduling (ICAPS-2004) – International Planning Competition. (2004) 21–23
16. Coles, A., Smith, A.: Marvin: Macro actions from reduced versions of the instance. In: Proc. Int. Conf. on Automated Planning and Scheduling (ICAPS-2004) – International Planning Competition. (2004) 24–26
17. Chen, Y., Hsu, C., Wah, B.: SGPlan: Subgoal partitioning and resolution in planning. In: Proc. Int. Conf. on Automated Planning and Scheduling (ICAPS-2004) – International Planning Competition. (2004) 30–32
18. Vidal, V.: A lookahead strategy for heuristic search planning. In: Proc. Int. Conf. on Automated Planning and Scheduling (ICAPS-2004). (2004) 150–159
19. Zhu, L., Givan, R.: Heuristic planning via roadmap deduction. In: Proc. Int. Conf. on Automated Planning and Scheduling (ICAPS-2004) – International Planning Competition. (2004) 64–66
20. Zhou, R., Hansen, E.: BFHSP: a breadth-first heuristic search planner. In: Proc. Int. Conf. on Automated Planning and Scheduling (ICAPS-2004) – International Planning Competition. (2004) 61–63
21. Haslum, P.: TP4'04 and HSP$^*_a$. In: Proc. Int. Conf. on Automated Planning and Scheduling (ICAPS-2004) – International Planning Competition. (2004) 38–40
22. Helmert, M., Richter, S.: Fast downward making use of causal dependencies in the problem representation. In: Proc. Int. Conf. on Automated Planning and Scheduling (ICAPS-2004) – International Planning Competition. (2004) 41–43
23. Sanchez, J., Tang, M., Mali, A.: P-MEP: Parallel more expressive planner. In: Proc. Int. Conf. on Automated Planning and Scheduling (ICAPS-2004) – International Planning Competition. (2004) 53–55
24. Botea, A., Enzenberger, M., Muller, M., Schaeffer, J.: Macro-FF. In: Proc. Int. Conf. on Automated Planning and Scheduling (ICAPS-2004) – International Planning Competition. (2004) 15–17
25. McDermott, D.: The optop planner. In: Proc. Int. Conf. on Automated Planning and Scheduling (ICAPS-2004) – International Planning Competition. (2004) 48–50
26. Porteous, J., Sebastia, L., Hoffmann, J.: On the extraction, ordering, and usage of landmarks in planning. In Cesta, A., Borrajo, D., eds.: Proc. European Conference on Planning (ECP-2001). (2001) 37–48

27. Garrido, A., Long, D.: Planning with numeric variables in multiobjective planning. In Saitta, L., ed.: Proc. European Conference on AI (ECAI-2004), Amsterdam, IOS Press (2004) 662–666
28. Fox, M., Long, D.: The automatic inference of state invariants in TIM. Journal of Artificial Intelligence Research **9** (1998) 367–421
29. Bäckström, C.: Computational aspects of reordering plans. Journal of Artificial Intelligence Research **9** (1998) 99–137
30. Sanchez Nigenda, R., Kambhampati, S.: $AltAlt^p$: Online parallelization of plans with heuristic state search. Journal of Artificial Intelligence **19** (2003) 631–657
31. Zimmerman, T., Kambhampati, S.: Generating parallel plans satisfying multiple criteria in anytime fashion. In: Proc. Workshop on Planning and Scheduling with Multiple Criteria (AIPS-2002). (2002) 56–66