Deordering and Numeric Macro Actions for Plan Repair

Enrico Scala  
Research School of Computer Science  
Australian National University  
enrico.scala@anu.edu.au

Pietro Torasso  
Dipartimento di Informatica  
Università’ degli Studi di Torino  
torasso@di.unito.it

Abstract
The paper faces the problem of plan repair in presence of numeric information, by providing a new method for the intelligent selection of numeric macro actions. The method relies on a generalization of deordering, extended with new conditions accounting for dependencies and threats implied by the numeric components. The deordering is used as a means to infer (hopefully) minimal ordering constraints then used to extract independent and informative macro actions. Each macro aims at compactly representing a sub-solution for the overall planning problem. To verify the feasibility of the approach, the paper reports experiments in various domains from the International Planning Competition. Results show (i) the competitiveness of the strategy in terms of coverage, time and quality of the resulting plans wrt current approaches, and (ii) the actual independence from the planner employed.

1 Introduction
Planning for real world systems requires the ability of reacting to unexpected contingencies in a timely fashion ([Ghallab et al., 2014]). While some uncertainty can be anticipated offline when there is enough knowledge about the possible conditions and evolutions of the world (e.g., probabilistic planning [Hoffmann and Brafman, 2006]), in many cases such a knowledge might be not at disposal (e.g., user requests may vary in a way that is not predictable a-priori); for this reason, it is necessary to implement deliberation not only before but also during the execution of the plan.

In this context several works ([Gerevini and Serina, 2010], [van der Krogt and de Weerdt, 2005], [Garrido et al., 2010], [Brenner and Nebel, 2009]) have faced the problem by proposing strategies for efficient plan repair. Despite contrasting results in the general case ([Nebel and Koehler, 1995]), it has been shown that in practice repairing a plan is much faster than replanning from scratch ([Muñoz-Avila and Cox, 2008]). Unfortunately, most attention has been posed just on propositional aspects, while real world applications have to deal with quantitative aspects as well (e.g constraints on continuous resources) which can be captured by using numeric fluents ([Fox and Long, 2003]); in order to be actually executable, plans have not only to be propositionally valid, but also consistent in terms of specific numeric trajectories.

In this work we tackle the problem of plan repair with numeric information extending a research line based on the adoption of numeric macro action ([Scala, 2014]). The basic idea is that the repair process can be enhanced by exploiting previous knowledge directly extracted from the plan in execution. As shown in [Scala, 2014], such a knowledge can be compactly represented in terms of (numeric) macro actions, then compiled as regular instances of actions and provided in input to any planner in an extended domain representation. A huge number of possible macro actions could be extracted: an intelligent strategy has to select just fruitful macro actions in order to deal with the Utility Problem ([Minton, 1990]).

Following this line of research, this paper provides a new method for the selection of numeric macro actions. The main idea is to try to find reasonable self-contained numeric macro actions, i.e., actions that do not depend on each other. To achieve this objective the paper extends deordering ([Bäckström, 1998]) for dealing with both numeric and propositional information. Deordering has proved to be beneficial in many other contexts of automated planning, for instance as a means to enhance the ability of absorbing unexpected situations ([Muise et al., 2011]) or to improve the quality of the computed solutions (e.g., [Siddiqui and Haslum, 2013]); the extension to the numeric case extends the applicability to a larger set of scenarios in which dealing with numbers cannot be neglected.

In particular, the paper provides conditions for establishing which are the ordering constraints that must be preserved or can be safely removed (without endangering the numerical validity of the plan). Then, the paper shows how the deordered plan can be used in the context of plan-repair for building numeric macro actions to be used in addition to regular operators. The approach is aimed at being domain and planner independent; to show its effectiveness we experiment on a set of domains using as off the shelf planners both Colin [Coles et al., 2012] and Metric-FF [Hoffmann, 2003].

The paper starts by introducing the formal framework of reference (Section 2), then the focus is on the deordering (Section 3), and on its exploitation (Section 4). Section 5 and 6 report experiments and discuss related works.
2 Background

The paper builds up current research on automated planning with numeric fluents. We assume the reader is familiar with the PDDL language\(^1\) and its semantic.

2.1 Basic Notions

Consistently with many works in the planning literature ([Fox and Long, 2003], [Ghallab et al., 2014]), we represent the world with a finite set of objects upon which we allow to express numeric fluents and propositional predicates. The universe of possible predicates is referred by \(F\) (propositional) and \(X\) (numeric). A world state \(s\) is the tuple \(\langle F(s), X(s)\rangle\); it specifies (i) the propositional predicates holding in \(s\), and (ii) a real value for each of the numeric fluents.

Definition 1 (Numeric Action). An action \(a\) is a pair \((\text{pre}(a), \text{eff}(a))\) where:

- \(\text{pre}(a)\) is the precondition set of a defining its applicability conditions. \(\text{pre}(a)\) is a conjunction of both propositional and numeric conditions. A propositional condition is a predicate \(\in F\), while a numeric condition is a comparison \(\langle \exp, \{<, \leq, =, \geq, >\}, \exp'\rangle\)

- \(\text{eff}(a)\) defines the effects of executing \(a\). It includes:
  - an add and a delete set of objects from \(F\) referred with \(\text{eff}_{\text{add}}(a)\) and \(\text{eff}_{\text{del}}(a)\)
  - a set of numeric operators \(\langle \text{eff}_{\text{num}}(a)\rangle\). An operator is formalized by the tuple \(\langle f, \text{op}, \exp \rangle\) where \(f\) is a fluent from \(X\) and \(\text{op}\) is an element from \(\{+,-,\cdot,=\}\).

\(\text{Exp}\) and \(\exp'\) are arithmetical linear expression of numeric fluents.

Consistently with works on numeric planning ([Hoffmann, 2003],[Coles et al., 2012],[Gerevini et al., 2008]), we will say that an action is applicable in a state \(s\) whenever its precondition satisfies \(s\) both in propositional and numeric terms. If \(s \models \text{pre}(a)\) then its application produces a new state \(s' = s[a]\) as follows. \(s'\) is set to \(s\); each atom in \(\text{eff}_{\text{add}}(a)\) is set to true while each atom in \(\text{eff}_{\text{del}}(a)\) is negated. The numeric part of the state is modified according to operators in \(\text{eff}_{\text{num}}(a)\). Let \(\text{op}\) be one of such operators, \(f\) reported in \(\text{op}\) is modified according to \(\text{op}\) while \(\exp\) is evaluated in \(s\).

Definition 2 (Numeric Planning Task). A numeric planning task \(\Pi\) is the tuple \(\langle A, s_0, G\rangle\) where \(A\) is a set of numeric actions, \(s_0\) is the initial state and \(G\) is a set of goals. \(\Pi\) is the task of finding a sequence of actions leading the state \(s_0\) to a state \(s_n\) satisfying \(G\). As action precondition, \(G\) may contain numeric and/or propositional conditions.

A solution plan \(\pi = \{a_0, ... , a_{n-1}\}\) is valid iff each action is applicable in the state produced by the previous action, i.e., \(s_0 \models \text{pre}(a_0), s_0[a_0] \models \text{pre}(a_1), ..., s_0[a_0,...,a_{n-2}] \models \text{pre}(a_{n-1})\), and then \(s_0[\pi] \models G\).

2.2 Plan Repair and Macro Actions

In a dynamic context an initially valid plan can become invalid at some point of the execution because of the combination of number of unexpected contingencies (exogenous events, erroneous assumptions, goal revisions). So the agent has to react to these conditions in order to achieve (possibly revised) goals despite the encountered discrepancies. In this paper in particular we are interested in understanding how it is possible to repair the plan. Formally:

Definition 3 (Numeric Plan Repair Problem from [Scala, 2014],[Fox et al., 2006]). A numeric plan repair problem \(\Psi\) is the tuple \(\langle A, s_i, G', \pi_i\rangle\) where \(s_i\) is the state observed after the execution of the \((i-1)\)-th action from the plan \(\pi\), \(G'\) is a new set of goals\(^2\), \(A\) is the universe of actions we are considering, and \(\pi_i\) is the suffix of the plan \(\pi\) still to be executed. A solution for \(\Psi\) is a plan \(\pi'\) from \(A\) bringing the state \(s_i\) to a state satisfying the conditions expressed in \(G'\).

Recently, it has been noticed that plan repair can be approached in a planner independent way by extrapolating knowledge from the previous plan in form of numeric macro actions ([Scala, 2014]). Those macros represent sub-plans that can be useful for repairing the plan, of course under the assumption that discrepancies between observations and expectations are limited, as well as changes in the goals set.

Basically a numeric macro action is a compact representation of a sequence of actions where preconditions and effects respectively represent (i) the sufficient and necessary conditions for executing that plan and (ii) the cumulative effects obtained after the execution of that sequence of actions. In the approach pursued by [Scala, 2014], this macro actions extraction is lead by the flaws of the previous plan and by the threats towards the goal conditions.

In this work we investigate a more sophisticated approach for the selection process. The main idea of our work is to extract macro actions that represent self contained components, i.e. sequence of actions that do not interact with each other, and can be executed independently.

Figure 1 describes the high level idea of this work. The main contribution is played by the two components in the MADE (Macro Actions via DEordering) module: the deorderer and domain enhancer components. The deorderer is responsible for inferring the actual action dependencies on the basis of the current plan of actions and the current situations (described in the problem instance). Then the domain enhancer exploits such a knowledge to build macro actions representing self-contained/independent components. Next section focuses on the way the deordering is implemented to manage the numeric components at hand.

3 Deordering Numeric Plans

The task of deordering a plan consists in finding the minimal set of orderings among actions such that any of its linearizations is valid [Bäckström, 1998]. In particular, given a numeric planning problem \(\Pi\), we are interested in finding the deorder of a total ordered plan solving \(\Pi\).

\(^1\)For details, have a look at [Fox and Long, 2003].

\(^2\)\(G'\) represents a variation w.r.t. the initial goal, where some predicate/comparison could be added/removed.
partial order is NP-hard, (Bäckström, 1998) does not suffice to infer action dependencies in the numeric context.

As also pointed out in [Gerevini et al., 2008], the support of an action might not depend on the execution of a single action, but in a number of situations it may depend on a set of actions (ordered in a certain way). As reported in Def. 1 the impact of the execution of an action on a variable is a function of an expression whose evaluation may involve all the fluents of the state where the action is applied.

In order to capture this aspect, we introduce the notion of numeric dependency set and its relation with respect to numeric conditions. Formally:

**Definition 4 (Numeric Dependency Set (NDS)).** A Numeric Dependency Set (NDS) of a condition $c$ is a set of actions such that $\forall s \exists O$ an ordering $O$ such that the execution of NDS using $O$ starting from $s$ satisfies $c$. We will say that, given an action $a$ and a NDS $K$, the action $a$ “numerically” depends on $K$, whenever $a$ has a condition $c$ having $K$ as NDS.

From the definition above we can deduce that:

**Proposition 1.** Let $a, \prec$ be a valid plan where $\prec$ defines a total order among actions in $A$, each action $a \in A$ has at least a NDS with the following characteristic:

- $\forall b \in NDS$ then $b \prec a$

The proof of the proposition is a direct consequence of the validity of a plan. In fact there is a trivial valid NDS for each condition $c$ for any action in $A$. This NDS corresponds to the plan prefix formed by the init action till the action strictly precedent to $a$. The init action is a means to simulate the whole initial status; for the numeric case this means that the action contains an assign operator for each numeric fluent of the problem.

In most cases there could be several NDSs. Therefore, since we are interested in minimising orderings among actions, the problem is how to look for the smallest NDS.

The problem involves a combinatorial number of sequences of actions to be taken into account. A NDS for an action $a$ can be in fact any combination of actions, in any order. While this dependency in the classical case can be found simply by looking for one of the achievers for the condition of $a$, the numerical case requires additional care.

In order to handle the problem in an efficient way, we implemented a hill climbing approach. Let $b$ be an action from $\pi$ and $S = \{a : a \in \pi$ and $a \prec b\}$, the algorithm performs a local search in the space defined by subsets of $S$, with the aim of finding all the (hopefully minimal) dependencies of $b$. For each numeric condition $c$ in $pre_{num}(b)$, the algorithm greedily adds new actions to the NDS of $c$ whose contribution decreases the distance from the satisfaction of the constraint. The contribution is computed by executing NDS actions from an empty state, using the order imposed by the input plan.

We consider only NDSs that are not invalidated by the execution of actions ordered between the first action of the NDS and the action for which the NDS has been built. The check is done by simulating the state resulting from the application of refuel action multiple times. A naive application of the producer/consumer model proposed for the classical setting (see [Bäckström, 1998]) does not suffice to infer action dependencies in the numeric context.

Figure 1: Overall sketch of the approach
of the actions from NDS and all the intermediate actions (according to the ordering provided by the input plan).

At the end of the process, we can deduce some sufficient ordering between actions. As matter of facts, if an action a belongs to some NDS for an action b, then a ≺ b. So we have the ordering constraints that must be considered.

If the plan in input is valid for the numeric planning problem, NDSs are found in polynomial time as the procedure must be repeated "just" as many times as there are conditions in the plan, and for each step the search does not backtrack to a previous decision. Completeness of hill climbing in this case is guaranteed by the fact that in the worst case it finds the trivial NDS entailed by the input plan.

The situation is more complicated if the plan presents open orderings which are valid.

In this section we see how it is possible to infer whether an ordering constraint can be removed. Algorithm 1 summarizes the main steps involved.

**Algorithm 1: Orderings Removal**

```plaintext
for (a,b) ∈ ≺* do
    preserve the constraint if at least one of the following holds:
    - a is pseudo init or b is a pseudo goal;
    ** Propositional Part **
    - ∃ (a,p,b) ∈ PVS for some p;
    - ∃ (c,p,a) ∈ PVS and p ∈ del(b);
    - ∃ (b,p,c) ∈ PVS and p ∈ del(a);
    ** Numeric Part **
    - ∃ (NDS,cmp,b) ∈ NVS, a ∈ NDS ;
    - ∃ (NDS,cmp,a) ∈ NVS and b might prevent cmp;
    - ∃ (NDS,cmp,c) ∈ NVS, b ∈ N DS and a might prevent cmp;
    - ∃ (NDS,cmp,c') ∈ NVS and a, b ∈ NDS;
```

The propositional step is part of the KK algorithm, hereby reported for clarity reasons. The difficulty here is in establishing when the removal of an ordering between two actions threatens the satisfaction of some comparison (see "Numeric Part"). In the pseudo code the tuple (NDS,cmp,a) represents the set any action numerically depends on. cmp is the numeric condition causing the dependency.

It can be shown that a sufficient condition for guaranteeing that an action is not a threat for a comparison can be inferred by reasoning on the numeric fluents influenced/required by the actions at hand. More formally:

**Proposition 2.** Given an action a and a comparison cmp. a does not threaten cmp if:
- a does not indirectly influence fluents involved in cmp
- cmp eff(a) is dominated by cmp

**Proof sketch.** An action a does not indirectly influence fluents in cmp when a’s operators do not contain any indirect change to variables involved in cmp. For cmp eff(a) we intend the regression version of the comparison where each fluent is substituted with the numeric expression implied by eff(a). A comparison c’ is dominated by another comparison c” whenever the validity region entailed by c’ contains the one entailed by c”. In that case in fact it holds that: s |= c’ ⇒⇒ s |= c. The proof can then proceed by absurd. If we assume in fact that the action may threaten the comparison this means that if we execute the action just before the evaluation of the constraint, the set of states from which the comparison is satisfiable can be smaller than the one from which it was satisfiable before. However, this contradicts the dominance hypothesis in that cmp eff(a) is less strict then cmp. Therefore, the action cannot threaten the comparison.

The first hypothesis in the proposition assures that the regression captures all the implications of action a; in fact indirect effects (i.e., changes to variables on which other variables involved in cmp depend) are not captured by the regression.

At the end of this process we hence know a possible deordering of the plan given as input. Next section explains how this information is used to build macro actions.

## 4 Select Macros via Deordered Flawed Plans

Once all the dependencies among actions within the plan have been computed, and hence a (hopefully more) relaxed partial order is singled out, this information is exploited to build macro actions.

As a first step we collect all the objects within the problem formulation and we transform those objects in constants. The constants are used in the domain definition in order to make possible to build instances of operators. In the context of plan repair macro actions are grounded, as a difference to macro actions used in learning based approaches (IChra et al., 2013)). This is a very important characteristic in that numeric macro actions does not suffer from the combinatorial explosion of cases that can happen when building macro action schema, because of the grounding process.

As a second step we run a connectivity analysis by inspecting the Directed Acyclic Graph entailed by the partial order. Each resulting connected component is used to build the associated macro action, by following the ordering imposed by the graph itself.

Actually, the resulting macro is obtained by performing a further splitting process. The graph is indeed cut by looking at the open conditions (caused by the discrepancies between the problem for which the plan is generated and the

---

3 The regression hereby performed is the same employed in [Scala, 2014] to compute preconditions of macro action.
actual problem formulation depending on the actual current observed state) and the goal threats. This is basically the strategy adopted by CLMA in [Scala, 2014], here extended for each connected component found. In our approach, the splitting points are computed during the deordering process. It is in fact sufficient to keep track of each condition that cannot be satisfied. These macro actions are used to enhanced the domain as shown in Fig. 1.

### 4.1 Example

As an example of the entire process accomplished by MADE, we have extrapolated a simple problem instance from the Zenotravel domain and performed some analysis all along the deordering and macro actions construction process. Figure 2a shows the DAG implied by the total ordered plan provided as a solution by the Metric-ff planning system. The DAG reports not only the explicit orderings, but also all the implicit ones computed performing the transitive closure of the order relation. As you can see, without any form of reasoning, it is quite inefficient to reason on this formulation as it may contain several ordering constraints which are actually not necessary. Figure 2b reports the DAG after the deordering process. As it is easy to see from the picture, the deordering is able to minimize quite an interesting amount of orderings; moreover, the numeric extension captures also the dependencies along the numeric dimension of the problem. The refuel action is in fact ordered before some successive fly actions, whose execution will not be possible with the amount of fuel provided from the initial state. More important the relaxed ordering formulation, as shown in Figure 2c, allows to isolate which are the part of the plan which are independent each other. Then, after the removal of the init and the goal pseudo actions we have the two separated components, upon which MADE builds the macro actions set. Each macro is built taking a possible linearization of the set of actions belonging to the entailed independent components.

In order to take into account the actual discrepancies from the new encountered planning problem, each resulting macro action is inspected and split whenever there is a node inside a connected component representing an action with at least a numeric or a propositional condition open. This is detected by the same deordering process, as a result of the hill climbing strategy (in that case the NDS for an action is empty).

For this problem instance, the CLMA strategy would have not considered all the action dependencies, so different macro actions are constructed looking at the encountered discrepancy. Let us assume that the actual discrepancy would have referred to the first connected component (the one on the left). Since CLMA has no knowledge of the independency w.r.t. the second component (the one on the right), the result would have consisted just two macro actions, for which at least one of them involves actions from the second component. This has two negative effects: on the one hand it could happen that the two macro actions will never be exploited because of the impossibility to recover from the inconsistency; on the other hand, if the macro is employed, the need of some precondition entailed by the fusion of that macro would imply the presence of other spurious actions whose involvement is actually not necessary. As a difference MADE computes three actions out of the deordered plan. The first is the one referring to the second connected component (which is already executable from the initial state), and other two actions computed splitting the component according to the CLIMA rule.

### 5 Experimental Evaluation

In order to evaluate the benefits (and possible drawbacks) of the approach described in the paper we have tested different system architectures in a number of domains. In particular this section compares the performance of a planner P using the DME (domain model enhanced) with the corresponding planner P using just the basic domain model. We are interested in evaluating whether there is an advantage in adopting the strategy MADE over the strategy CLMA introduced in [Scala, 2014]. The experiments make use of two planners: Metric-ff and Colin. Hence we have six architectures to evaluate: Metric-ff-MADE, Metric-ff-CLMA, Metric-ff-basic.
Colin-MADE, Colin-CLMA, Colin-basic. Suffixes denote the particular selection strategy employed. Basic refers to the architecture where no macros are used. For the sake of comparison, we also report results obtained via LPG-ADAPT which is the state of the art system for plan adaptation and is based on quite different principles wrt a macro action approach.

The experimental setting. The system has been tested on 5 numeric domains of the International Planning Competition (IPC): DriverLog, Rovers, Depots, Satellite and Zeno-Travel. To challenge the system in the management of resource constraints, we preferred domains where actions are preconditioned in numeric terms (in addition to propositional conditions). We have added a sixth domain (denoted as Zeno-TravelPlus) which is a variant of ZenoTravel where the refuel action has the additional constraints that it can be performed just in specific airports. To generate several instances of plan repair problems we used an approach similar to the one reported in [Fox et al., 2006] and [Scala, 2014].

In particular, for each considered domain, the starting problems are the 10 most difficult problems of the benchmarks planning suite. For each of them, a starting plan has been generated (by using LPG), and a set of variant problems have been collected. Each variant differs from the starting problem problem because of the addition and/or removal of up to 3 initial state information and 3 goals. For each domain we collected 100 test cases where for each case the injected change makes the input plan no more valid, so that for each test case a not trivial plan repair step has to be performed.

The performances of the systems are compared according to: coverage represents the number of test cases that a given system is able to solve within a timeout; time represents the CPU time taken by the system in solving the test case. Plan-length represents the length (in terms of original atomic actions in the domain model) of the repair plan which is produced by the system for solving the test case. This parameter is quite relevant since in principle the plan produced via macro actions could involve many atomic actions that are not relevant for solving the problem at hand (useless actions). We have also collected experimental results for a parameter somewhat related to plan stability. In particular we have computed the plan distance between the plan obtained as repair plan with respect to the plan given for the starting problem. This parameter is quite relevant since in principle the plan produced via macro actions could involve many atomic actions that are not relevant for solving the problem at hand (useless actions). We have also collected experimental results for a parameter somewhat related to plan stability. In particular we have computed the plan distance between the plan obtained as repair plan with respect to the plan given for the starting problem. The notion of plan distance used in the experiments has been introduced in [Fox et al., 2006]. The performances of the systems are measured according to the International Planning Competition metrics. Given a parameter p (distance or plan-length), the score of a case for the system s in a set of tested systems S is defined by means of \( \frac{bestValue(p,S)}{value(p,s)} \). For the time score, let \( T^* \) be the minimum time required by any planner, the formula \( 1/(1 + \log_{10}(T/T^*)) \) is used to evaluate the performance of a system which spent T sec to solve the case. Cases solved in less than 1s take the maximum score 1. Coverage scores 1 for solved case, 0 otherwise. The total score for a domain is the sum of scores obtained for each case in that domain.

The experimental results reported in Table 1 show that MADE always outperforms the basic strategy in terms of coverage, time and plan-length. These results make clear that the addition of macro actions is very beneficial since it not only improves the coverage score, but also time score (despite the addition of the preprocessing time). Even more surprising is the result of plan-length score which shows that the macro actions are not responsible for adding useless actions in the plan. It is worth noting that MADE does not create a large number of macro actions to be added to the model, also if in general this depends on the specific case, and in particular is strictly related to the length of the plan and the number of interactions between actions. We measured that, on the average, MADE adds 4.7 macro actions into the extended model representation. The addition of a limited number of selected macro actions experimentally turned out to be a good compromise between the undeniable increase of the branch-

<table>
<thead>
<tr>
<th>Domain</th>
<th>Made-FF</th>
<th>Basic</th>
<th>Made-FF</th>
<th>Basic</th>
</tr>
</thead>
<tbody>
<tr>
<td>DriverLog</td>
<td>67</td>
<td>65</td>
<td>35</td>
<td>27</td>
</tr>
<tr>
<td>Rovers</td>
<td>78,12</td>
<td>70</td>
<td>40,5</td>
<td>36,2</td>
</tr>
<tr>
<td>Depots</td>
<td>98</td>
<td>97</td>
<td>76</td>
<td>66,7</td>
</tr>
<tr>
<td>Satellite</td>
<td>95,71</td>
<td>90</td>
<td>73,37</td>
<td>89,38</td>
</tr>
<tr>
<td>Zeno-Travel</td>
<td>50</td>
<td>47,5</td>
<td>34,3</td>
<td>38,5</td>
</tr>
<tr>
<td>Total</td>
<td>463</td>
<td>381</td>
<td>270</td>
<td>252</td>
</tr>
</tbody>
</table>

Table 1: Results of experiments expressed using IPC scores. C, T, P, D are shorts for Coverage, Time, Plan-length and Distance score. Timeout in these experiments is set to 20secs. Bold is used to highlight the advantage of the employed macro action selection strategy.
The benefits of using macro actions is also shown by the performance of CLMA strategy that provides better results than the basic strategy in all the domains as concerns covering, time and plan length. A challenging comparison is between MADE and CLMA: in almost all the domains and in the global score, MADE got much better scores as concerns coverage, time and plan length (for example the differences in the Rover domain are actually huge). The result is unexpected as concerns the time since MADE has to perform not trivial steps before starting the synthesis of macro actions.

The ranking \textit{MADE} \textgreater \textit{CLMA} \textgreater \textit{Basic} is the same both considering Metric-FF and Colin even if the underlying planner has a significant impact on the global performance (in particular Colin in some domains such as Depots and ZenoTravel has performances much lower than Metric-FF). This is a further evidence that the addition of macro actions is beneficial independently on the specific planner adopted for producing the plan repair. As concerns the distance score, the experiments confirm what is expected: the basic strategy produces plans which are more distant from the original one than MADE and CLMA do. For this parameter, the comparison between MADE and CLMA shows that there is no clear winner, while globally CLMA has slight better performance. In Table 1, we have listed also the performances obtained by LPG-ADAPT for a comparison with a system which directly works on the original plan for producing the repair plan. We expected that LPG-ADAPT performs better than strategies exploiting macro actions, not only as concern time (all the CPU time can be exploited for plan adaptation) but also coverage (the structure of the original plan is a strong guide for obtaining a repair plan). The experiments confirmed this hypothesis, but at least in the ZenoTravel domain (and its variant) the performances of LPG-ADAPT have been much worse than the one obtained by Metric-ff-MADE.

The timeout $\Delta$ is an important parameter in the experimental setting: thus, we have performed an extensive evaluation on its impact on the coverage. In Figure 3 we report the global coverage on the 600 test cases (100 for each domain) by setting $\Delta$ from 1 to 100 seconds. It is easy to see that the ranking \textit{MADE} \textgreater \textit{CLMA} \textgreater \textit{Basic} holds for all the values of $\Delta$. More interesting, there is a very limited increase in coverage when $\Delta$ increases. In particular, for Metric-FF-MADE the coverage when $\Delta$ is set to 20s. is almost the same when $\Delta$ is set to 100s. It is also worth noting that for small values of CPU timeout (roughly 40s.) the coverage got by Metric-ff-MADE is even larger than the one of LPG-ADAPT.

By looking at the Raw-Data of our experiments we noticed that a very large number of test cases have been solved in less than 1s. A large amount of CPU time (e.g. $> 20$ s.) is actually needed just in the ZenoTravel domain (but in this domain problems required very long plans; in our test cases the average length is 213,87 and the longest plan involves 320 actions). Also in Driverlog the plans are very long (up to 236 actions). A final remark concerns the CPU time taken by deordering and extraction of macro action in Metric-ff-MADE. In our experiments we measured the average time taken by the preprocessing step over the total time taken for solving each single test case, and we have seen that this time takes up to the 77% in the Rover Domain (explaining the huge success of MADE over CLMA) while is just the 15% in the ZenoTravelPlus. Because of the length of the plans given as input, in the ZenoTravel the absolute average time spent for the deordering is roughly 3 secs.

6 Discussion and Conclusions

The paper addresses the problem of plan repair for tasks characterized by the presence of numeric conditions, in addition to propositional ones. In particular the paper presents a new method for an accurate selection of numeric macro actions from the plan in execution.

Macro actions have been exploited in different contexts ([Chrpa \textit{et al.}, 2013],[Botea \textit{et al.}, 2005],[Coles and Smith, 2007],[Scala, 2014],[Korf, 1985]) as a means to speed up the resolution process by providing shortcuts in the search space. Whenever a macro action is employed, a state space planner can jump from states which are not explicitly connected. In the paper we use the same structure of macros presented in [Scala, 2014], but as a difference we use quite a different selection strategy. At the basis of the selection the paper presents an extended version of deordering to deal with tasks involving numeric information. In particular we have introduced (i) the notion of numeric dependency set in order to explain causality also in the context of numeric planning (ii) a sufficient condition for the safe removal of ordering constraints considering numeric threats.

Deordering/reordering has been thoroughly investigated in previous works ([Bäckström, 1998],[Kambhampati, 1994],[Muise \textit{et al.}, 2012]). In this paper we have extended the mechanism to infer not only propositional dependencies but also numeric ones. At the basis of this extension the observation that the executability of the action may depend on the particular schedule of a set of other previous actions. In the context of plan repair, works have been mainly focused on propositional or temporal discrepancies([Garrido \textit{et al.}, 2010],[van der Krogt and de Weerdt, 2005]; [Cushing and Kambhampati, 2005];[Conrad and Williams, 2011]).
and Williams, 2014]). [Fox et al., 2006] deals with the problem of plan repair with numeric information extending the LPG system to produce high stability plans. Our work tackles the problem from a different perspective, which is a planner independent one.

Despite reported conditions are just sufficient, experimental results show the benefits of the entailed macro action selection strategy. MADE (Macro Action and DEOrdering), wrt CLMA ([Scala, 2014]). Looking at our experiments we noticed that in many cases MADE finds decompositions directly from the implicit multi-agent nature of the submitted problem. The strategy turned out to be even very competitive (for lower timeouts) with specialized plan repair system ([Fox et al., 2006]). In line with the work of [Muise et al., 2011], an immediate future work is in measuring the flexibility of the partial ordered plans computed as a result of the deordering process. Its least commitment nature can be in fact very useful in absorbing unexpected contingencies, without the necessity of invoking a (possibly expensive) replanning step. An interesting aspect that can be also investigated as future work is whether it is possible to exploit macro actions for considering temporal aspects and concurrency, compulsory in some situation ([Cushing et al., 2007]).

Acknowledgements. This work was supported in part by ARC project DP140104219, Robust AI Planning for Hybrid Systems, and "Intelligent agents for autonomous system" - Dipartimneto di Informatica - Università di Torino.

References


[Muise et al., 2011] Christian Muise, Sheila A McIlraith, and J Christopher Beck. Monitoring the execution of


