Numeric Kernel for Reasoning about Plans Involving Numeric Fluents

Numeric Kernel for Reasoning about Plans Involving Numeric Fluents

Enrico Scala

Dipartimento di Informatica, Universita' di Torino, Torino, Italy
scala@di.unito.it

Abstract. The paper proposes the notion of *numeric kernel* as a means for reasoning about plans involving numeric state variables, i.e. numeric fluents. A *numeric kernel* identifies the sufficient and necessary conditions that allow to directly - without any search and any propagation - assess whether a plan is valid in a specific world state. The notion generalizes the propositional kernels defined for the STRIPS language, to support domains involving numeric information as well. A regression method to build such kernels is reported, and its correctness is theoretically proved. To evaluate the numeric kernels contribution, we report two possible repair strategies that can be employed as a direct application of the numeric kernel properties. Results show the promise of the approach both from the computational point of view and in terms of plan quality.

1 Introduction

In the last decade of research in AI, an increasing amount of work has been devoted in extending the automated planning (and especially the classical paradigm) for dealing with real world problems ([1], [2], [3]). One of the shortcomings of the classical setting is the lack of the management of consumable and continuous resources as well as of the capability of reasoning about quantitative characteristics of the world. To this end, the *numeric fluent* notion and hence the *numeric planning* formalism have been introduced ([4], [2]). As an innovation w.r.t. the classical paradigm, in the numerical planning setting, plans must obey to particular resource profiles. This is achieved by allowing to express conditions and operations over the set of numeric variables of interest.

However, while efforts have been devoted for the problem of off-line plan generation ([1], [5], [2], [3]), few attention has been paid in studying (numeric) plan of actions for the on-line phase, making exception for the works dealing with the temporal dimension. In this context models as STP (Simple Temporal Problem) and DTP (Disjunctive Temporal Problem) have been adopted and some extensions have been proposed ([6] [7] [8] [9] [10]). However, also other continuous resources (e.g. energy, money and so forth) should be managed.

The main contribution of this paper is the notion and the mechanism for the construction of *numeric kernels*. A *numerical kernel* identifies the set of sufficient and necessary conditions allowing, for a given goal G and a plan π, to directly (without performing any search and any propagation) assess whether a

particular state of the system is consistent with G and π. The *numeric kernel* generalizes the propositional definition ([11]) for the *numeric* setting. Therefore, the extension allows the application of some properties studied in the classical setting for the numeric case, too. Such properties have been exploited (in the classical setting) in the plan execution context for improving the monitoring ([12]) and the repair ([13]).

As a complementary contribution, the paper presents a continual planning agent ([14]) and two effective plan repair strategies for dealing with numeric information. In particular, one of these two strategies shows that is possible to combine *numeric kernels* with the heuristic mechanisms developed for the numeric planning (e.g. [1],[2],[15]).

After a brief introduction on the formal framework of reference, the paper formalizes the notion of *numeric kernels* (section 2.2) and presents how such kernels can be constructed (section 3). Then, the work describes and (experimentally) evaluates the two repair strategies above (section 4 and 5).

2 Formal Framework

This section reports the reference planning formalism and then it formalizes the *numeric kernels* notion. We assume the reader is familiar to the PDDL-like language; for a thorough discussion see [4].

2.1 Basic Definitions

Definition 1 (World State). *A world state is built upon a set F of propositions and a set X of numeric variables. Thus a state s is a pair* $< F(s), X(s) >$*, where F(s) is the set of atoms that are true in s (Closed World Assumption) and X(s) is an assignment in* $\mathbb{Q}$ *for each numeric variable in X.*

Definition 2 (Numeric Action[1]). *Given F and X as defined above, a* numeric action *"a" is a pair* $< pre, eff >$ *where:*

- "pre" *is the applicability condition for "*a*"; it consists of:*
  - *a numeric part (*pre_{num}*), i.e. a set of comparisons of the form* $\{exp$ $\{<,\leq,=,\geq,>\}$ $exp'\}$.
  - *a propositional part (*pre_{prop}*), i.e. a set of propositions defined over* F.
- "eff" *is the effects set of* a*; it consists of:*
  - *a set of numeric operations (*eff_{num}*) of the form* $\{f,op,exp\}$*, where* $f \in X$ *is the* numeric fluent *affected by the operation, and op is one of* $\{+=,-=,=\}$.
  - *an "add" and a "delete" list (*eff^+ *and* eff^-*), which respectively formulate the propositions produced and deleted after the action execution*

Here, exp *and* exp' *are arithmetic expressions involving variables from* X*. An expression is recursively defined in terms of (i) a constant in* $\mathbb{Q}$ *(ii) a numeric fluent (iii) an arithmetical combination among* $\{+,*,/,-\}$ *of expressions*[2].

[1] For the sake of the explanation we refer to ground actions. However, in our implementation we support action schema as well.

[2] For computational reasons, several numeric planners request such expressions to be linear (e.g. [1],[3]). In our case, such a restriction is not necessary (see section 3).

An action a is said to be applicable in a state s iff its propositional and numeric preconditions are satisfied in s. Meaning that (i) $pre_{prop}(a) \subseteq$ F(s) and (ii) $pre_{num}(a)$ must be satisfied (in the arithmetical sense) by X(s).

Given a state s and a numeric action a, the application of a in s, identified by $s[a]$, (deterministically) produces a new state s' as follows. s' is initialized to be s; each atom present in $eff^+(a)$ is added to $F(s')$ (whether this is not already present); each atom present in $eff^-(a)$ is removed from $F(s')$; each numeric fluent f of the numeric operation $\{f$,op,exp$\}$ is modified according to the *exp* and the *op* involved. The state resulting from a non applicable action is undefined. An undefined world state does not satisfy any condition.

Definition 3 (Numeric plan). *Let I and G be a world state and a goal condition*[3]*, respectively, a* numeric plan $\pi = \{a_0, a_1, .., a_{n-1}\}$ *is a total ordered set of actions such that the execution of these actions (in the order defined by the plan) transforms the state I into a state I' where G is satisfied.*

Given the formulation above, we allow the access to a segment of the plan by subscripting the plan symbol. More precisely, $\pi_{i \to j}$ with $i < j$ identifies the subplan starting from the i-th till the j-th action. Moreover when the right bound is omitted the length of the plan is assumed, i.e. $\pi_i \equiv \pi_{i \to |\pi|}$. Finally, we identify by $s[\pi]$ the state produced executing π starting from s.

2.2 Numeric Kernel

When the plan has to be handled online, the presence of deviations from the nominal state (e.g., unexpected events, wrong assumptions made at planning time or actions that achieve different effects) may prevent the feasibility of the plan, and just checking the next action preconditions could not suffice to establish whether the plan still achieve the goals. Indeed, it is necessary to simulate the whole plan execution to predict if the goal is reachable via such a plan. More formally we can say that:

Definition 4 (Numeric i-th Plan Validity). *Let s be a world state and G a set of goal conditions, the sub-plan π_i is said to be* i-th *valid w.r.t. s and G iff $s[\pi_i]$ satisfies G.*

Since the PLANEX system ([11]) and more recently in [12], it has been noticed that the simulation step can be avoided by keeping trace of just a subset of (propositional) conditions; such conditions are also referred as the weakest preconditions of a plan.

However, the works in literature discussed so far just focused on the propositional fragment of the planning problem. Hence, to handle the numeric setting, we need to specify conditions even on the numeric part of the problem. That is:

Definition 5 (Numeric Kernel). *Let π be a numeric plan for achieving G, and K a set of (propositional and numeric) conditions built over F and X, K is*

[3] A goal condition has the same form of the applicability condition of a numeric action.

said to be a numeric kernel *of π iff it represents a set of sufficient and necessary conditions for the achievement of the goal G starting from s via π. That is, given a state s, $s[\pi]$ satisfies G iff s_{num} satisfies K_{num} and s_{prop} satisfies K_{prop}.*

A kernel is well defined when all the involved numeric comparisons are satisfiable by at least an assignment of values (e.g., a comparison $4 < 3$ is not allowed). This means that there must exist a state s, satisfying such kernel, for which *X(s)* gives a value for each numeric variable. An ill-defined kernel comes from a plan π and a goal G in which π cannot be a solution of any planning problem having G as goal.

By considering each suffix of the plan π, i.e. $\pi_0 = \{a_0, ..., a_{n-1}\}$, $\pi_1 = \{a_1, ..., a_{n-1}\}$, $\pi_2 = \{a_2, ..., a_{n-1}\}$,...,$\pi_{n-1} = \{a_{n-1}\}$ till the empty plan $\pi_n = \{\}$, it is possible to individuate an ordered set of *numeric kernels* where the i-th element of the set is the *numeric kernel* of π_i. It is worth noting that, by definition, the goal is a special kind of *numeric kernel* for the empty sub-plan.

Therefore, given a plan of size n we can say that:
- $s^0[\pi_0]$ satisfies G iff s^0 satisfies K^0
- $s^1[\pi_1]$ satisfies G iff s^1 satisfies K^1
- ...
- $K^n = G$ corresponding to the kernel for the empty plan π_n

where the superscript indicates the "time" index of interest. The resulting set of kernels will be denoted with $\mathbb{K}$, i.e. $\mathbb{K} = \{K^0,...,K^n\}$.

Given $\mathbb{K}$ defined as above, and the plan validity notion reported in Definition 4, it is possible to deduce that:

Proposition 1. *A plan π is valid at step i for a goal G iff s^i satisfies K^i, where*

- *s^i is the state observed before the execution of the subplan π_i*
- *K^i is the i-th kernel associated with π_i*

By observing this relation it is also possible to see that:

Proposition 2. *Let $\mathbb{K} = \{K^0,..,K^n\}$ be the kernel set associated with the plan $\pi = \{a_0, ..., a_{n-1}\}$, and s a world state, if there is a plan π' such that $s[\pi']$ satisfies K^i (with $0 \leq i \leq n$), then a plan from s to a state s' satisfying K^j (with $i \leq j \leq n$) exists, too.*

Basically, the proposition above provides a sufficient condition for the reachability of a kernel (included the goal) starting from a state satisfying such a condition. For instance, if it is possible to reach at least one of the kernel K via a given plan π'', a planning task, having s as initial state and G as goal, admits at least a solution, which is the one given by the concatenation of π'' and the suffix of the plan relative to K.

As we will see in section 3, the construction of the *numeric kernels* can be done just once, in a pre-processing phase. Once obtained the kernels set, thanks to Proposition 1, the validity checking process is very easy; it can be performed by simply substituting the numeric values of the state in each comparison appearing in the kernel, as well as, for the propositional part, it suffices to check if

each proposition is included in the current world state. For this reason one can infer whether the goal is still supported by the current state without analyzing the plan. Of course, in case the plan undergoes some adjustments, the set of kernels has to be recomputed. Moreover, thanks to Proposition 2, in case some inconsistency is detected, it could be not necessary to replanning from scratch. Indeed, it may suffice, firstly, achieving one of the kernel conditions, and then applying the remaining subplan. Both propositions are exploited in the continual planning agent of section 4.

3 Kernel Construction

Algorithm 1 reports the regression mechanism to build the kernel set.

Algorithm 1: Numeric Kernel Computation (NKC)

Input: $\pi = \{a_0,..,a_{n-1}\}$ - plan ; G - goal
Output: $\mathbb{K}$: an ordered set of numeric Kernels
1 $K^{|\pi|} = G$
2 **for** i=$|\pi|$-*1* to *0* **do**
3 $\quad K^i_{prop} = \{K^{i+1}_{prop} \setminus eff^+(a_i)\} \cup pre_{prop}(a_i)$
4 $\quad K^i_{num} = \{K^{i+1}_{num} \oplus eff_{num}(a_i)\} \cup pre_{num}(a_i)$

In particular, the procedure starts from the last kernel corresponding to the set of the goals (the propositions that must be achieved at the final state, given the comparisons on the involved numeric fluents, line 1), and produces each i-th kernel by performing two independent steps:
– (propositional fragment) - (i) removing the atoms provided by a_i (i.e. the add-list of the i-th action), (ii) adding the atoms required by a_i (i.e. the propositional preconditions of the i-th action), line 3.
– (numeric fragment) - combining the information involved in (i) the numeric part of the action model and (ii) the next *numeric kernel* (previously computed), keeping trace of the numeric action contribution.

The numeric part construction relies on the operator $\oplus$, which is a function that maps a set of comparisons C and a set of numeric action effects *eff* to a new set of comparisons C'. The operator assures that the new comparisons set will take in consideration the *future* effects of the action.

In particular, for each comparison in C the operator (see algorithm 2) performs a substitution of the numeric fluents involved in c_l and c_r[4], according to the effects reported in a. For instance, if the numeric effects set of a involves a fluent x with $(x+ = 5)$ and the comparison asserts that $x < 4$, then the outcome C' will be $x + 5 < 4$, i.e. $x < -1$. Of course, the action can affect many fluents involved in the previous comparison; therefore the substitution has to iterate over all the fluents involved in C with the effects described in the action model.

[4] c_l identifies the left part of the comparison while c_r the right one. Thus both c_l and c_r are arithmetical expressions over X.

Algorithm 2: $\oplus$

Input: eff: numeric effects of a ; C: a set of comparisons
Output: C': a new set of comparisons

```
C' = {}
foreach c ∈ C do
    c'_l = c_l.substitution(x_0...x_{m-1},eff_{x_0}...eff_{x_{m-1}})
    c'_r = c_r.substitution(x_0...x_{m-1},eff_{x_0}...eff_{x_{m-1}})
    C' = C' ∪ {c'}
```

In the algorithm, each eff_{x_i} is the numeric effect of an action, having x_i as affected numeric fluent.

As anticipated before, it is evident that the expression *exp* involved in a numeric effect $\{f, op, exp\}$ does not require to be linear; in fact, the mechanism does not regress to a specific state, but to a set of new conditions that substitute each fluent with the way such a fluent is altered.

For simplicity, let us introduce a small example, focusing on the numeric part of the model. Let us consider a "one action" plan $\pi = \{a_0\}$ and a goal G, where a_0 and G are numerically defined as follows:

$$\text{pre}(a_0) = \begin{cases} f_1 > 5 \\ f_2 < 4 \end{cases} \quad \text{eff}(a_0) = \begin{cases} f_1 = f_1 + 5 \\ f_2 = f_2 + 8 \end{cases} \quad \text{G} = \begin{cases} f_1 > 10 \\ f_2 < 4 \end{cases}$$

By following algorithm 1, we have $K^1 = G$ and (the numeric part of) K^0 created by means of $\{K^1_{num} \oplus eff_{num}(a_0)\} \cup pre_{num}(a_0)$. So in the first step, we have to keep trace of the action contribution, that is:

$$K^1 \oplus eff_{num}(a_0) = \begin{cases} f_1 + 5 > 10 \\ f_2 + 8 < 4 \end{cases}$$

Then, the *numeric kernel* is completed by joining the constraints defined by $\oplus$ with the constraints defined for *pre(a)*, i.e.:

$$(K^1 \oplus eff_{num}(a_0)) \cup pre_{num}(a_0) = \begin{cases} f_1 + 5 > 10 \\ f_2 + 8 < 4 \\ f_1 > 5 \\ f_2 < 4 \end{cases} = \begin{cases} f_1 > 5 \\ f_2 < -4 \end{cases}$$

It is easy to see that if we take an arbitrary (initial) state with f_1 and f_2 satisfying the arising comparisons (e.g. $f_1 = 6$ and $f_2 = -5$), and, if we apply to such a state the action a_0, we are sure that (i) the action is applicable, since $6 > 5$ and $-5 < -4$, and (ii) we will obtain a state s' where both $f_1 > 10$ and $f_2 < 4$ hold, namely goal conditions will be satisfied.

The example reported above describes a simple scenario where numeric fluents do not depend on each other. But this is not always the case. Theoretically,

in fact, an effect for the action can express that $f_1 = f_2$. Also this kind of representation is captured by the substitution performed by the algorithm 2. In general the algorithm will substitute each variable of the comparison with the way in which the variable is modified (line 3 and 4).

Let us conclude this section with the proof of the correctness of the algorithm 1. For the sake of explanation, the correctness proof focuses on the numeric aspect of the problem, so G, s, K and the plan π are analyzed as far as it is concerned by their numeric part.

Theorem 1. *Correctness. Given a plan π and a goal G the algorithm 1 finds a set of* numeric kernels *for π and G.*

Proof. The proof proceeds by induction on the length of the plan. The base case of our induction is when the plan is empty, i.e. $|\pi| = 0$; in such a case the algorithm terminates after one iteration with the last and the only *numeric kernel* of interest, i.e. $K^n \equiv K^0$ since $n = 0$. This kernel contains the same comparisons present in the goal; hence, given a state s, it follows that $s[\pi]$ satisfies G if and only if s satisfies K^0. In particular $s[\pi]$ corresponds to s and the only way of reaching the goal is to be a state that already satisfies the goal.

Inductive step. For inductive assumption we know that the *i-1* steps of the algorithm NKC (the iteration) have computed a set of i *numeric kernels* for a plan of length n; i.e. we have the set of kernels K = $\{K^{n-(i-1)}, K^{n-(i-2)}, ..., K^{n-1}, K^n\}$ which is in relation with each suffix of the plan, i.e. with $\pi_{n-(i-1)}$,$\pi_{n-(i-2)}$,...,π_{n-1}, π_n. At the i-th step, K^{n-i} is computed by combining the preconditions of the first action in π_{n-i}, i.e. a_{n-i} and the comparisons obtained in the previous step. For this reason, a state s satisfying K^{n-i} will be such that (i) the action a_{n-i} is applicable and (ii) the state resulting from the action application, i.e. $s[\{a_{n-i}\}]$, turns out to satisfy the $K^{n-(i-1)}$. This last in fact follows directly from the definition of $\oplus$. Indeed the operation keeps the conditions expressed in $K^{n-(i-1)}$, while considering the effects of a_{n-i} by means of the substitution mechanism. Having both (i) and (ii) we are sure that if s satisfies K^{n-i} , then $s[\pi_{n-i}]$ will satisfy the goal.

For the *necessary condition*, that is if $s[\pi_{n-i}]$ satisfies G then s satisfies K^{n-i}, we can proceed by absurd. Indeed, if s does not satisfy the conditions expressed in K^{n-i}, it means that either the first action of π_{n-i} (a_{n-i}) is not applicable or the state resulting from the application of such an action does not satisfy $K^{n-(i-1)}$. The latter is not possible for inductive assumption as $K^{n-(i-1)}$ is assumed to be a kernel, while for the former it is obviously impossible since if the action is not applicable then $s[\pi_{n-i}]$ will not satisfy the goal. This proves the contradiction.

4 Continual planning via Numeric Kernels

To validate and evaluate the relevance of the *numeric kernels* formulation, we implemented a continual planning agent able to handle domains with numeric fluents. The continual planning paradigm [14] allows the agent to interleave the

execution and the planning all along the task to execute. The idea is that the planning phase cannot be seen as a single one shot task. Instead, the agent should monitor the plan execution and replan or repair the plan when necessary.

In particular, when the current plan becomes inconsistent, it is important that the agent acts in a timely fashion to recover from the unexpected impasse. To this end, in the following, we propose two plan repair strategies which directly exploit the *numeric kernel* properties. For further details on how implementing the overall continual planning loop see [16].

4.1 Greedy Repair

The first implemented repair strategy is a direct application of the Proposition 2. Rather than abandoning the old plan for computing a solution completely from scratch, the greedy repair tries to restore the kernel conditions by performing a patch plan that connects the current state with a state that satisfies the expected kernel condition, at that step. Then the patch can be concatenated with the previous plan, forming a new plan of actions leading the agent to the goal conditions. We expect, in fact, that in some situation (i.e. where the deviation from the nominal trajectory is not so prominent), despite contrasting complexity results on this topic ([17]), it may be actually more convenient to adapt the old solution rather than replanning completely from scratch. However, there may be situations where the greedy nature of this strategy can be a waste of time, and, instead, there may be kernels all along the plan, that are actually easier to reach. To mitigate this behavior we present the kernel heuristic repair.

4.2 Kernel Heuristic Repair

As we have seen, each element of $\mathbb{K}$ identifies the conditions for a suffix of the plan to be valid. This means that, if the agent would have the capability of connecting the current state to one of such conditions, the relative suffix of the plan will bring the agent to reach the final goal.

Therefore, the idea is to search for the kernel that is the one actually *closer* (and hence hopefully simpler) to satisfy. However, the exact estimation of such distance is prohibitive as it would correspond in solving several instances of new planning tasks. For this reason, we combined the kernel set $\mathbb{K}$ with the numeric planning graph heuristic, developed in the context of numeric planning (e.g. in Metric-FF, [1], and Lpg-TD, [2]). In particular, the heuristic is the solution length of a relaxed version of the planning problem; the relaxation consists of considering the actions only in their positive effects[5].

In our context, the heuristic provides us with an estimation of the distance between the current state and the set of kernels $\mathbb{K}$; the resulting strategy is straightforward. Let s be the current world state at step i of the execution violating the condition expressed in the i-th kernel, the *kernel heuristic repair*

[5] The numeric planning graph heuristic extends the original definition for the classical paradigm (where actions are considered without their delete effects) to the domains involving numeric variables. For details see [1] and [2].

estimates the distance d(s,K_j) for each $K_j \in \mathbb{K}$ where $j \geq i$. Then the strategy picks the kernel which has the lower distance and will use this kernel as an intermediate point towards which performs the patch-plan. Having computed this new course of actions, as in the greedy repair, it suffices to concatenate the patch plan with the relative plan-suffix to obtain a valid plan from s to the goal G. The procedure is summarized by the algorithm 3.

Algorithm 3: Kernel_Heuristic_Repair

Input: π - numeric plan ; $\mathbb{K}$ - kernels set ; s - world_state
Output: the numeric plan modified

```
1 K* = best_kernel(s,𝕂);
2 if d(s,K*) ≠ ∞ then
3 |   π' = solve(s,K*);
4 |   if π' is not failure then
5 |   |   return π' ∪ suffix_of(K*)
6 return ∅
```

By comparing these two strategies we can see that, on the one hand, the greedy repair immediately commits towards a specific kernel without spending efforts to reason on the previous plan. On the other hand, the kernel heuristic repair has to spend some extra time to estimate what is the "best" kernel to take into account (i.e. the heuristic computation). We expect that, in the long run, the kernel heuristic repair could take advantage from this reasoning and hence outperforming the greedy approach in those cases for which making (or trying to make) a patch towards the current kernel is actually a useless time consumption.

5 Experimental Results

To evaluate the repair mechanisms proposed in the previous section, we tested the greedy repair and heuristic kernel repair on three domains from the third International Planning Competition (IPC3)[6]. The first is the ZenoTravel (ZT), the second is a harder version, where the refuel action is allowed only in presence of the refuel station (Hard ZT), and the third is the Rover domain.

For each domain, we used 11 problems from the IPC suite (the harder ones), and for each problem, we generated (off-line) a starting plan. To evaluate the plan repair strategies, we simulated the plan execution by injecting discrepancies on the way in which resources are consumed. For each case we injected 5 different amounts of noise; hence each case for our evaluation is identified by the tuple $< domain, problem, plan, noise >$. Totally we have 55 cases for each domain.

[6] http://www.plg.inf.uc3m.es/ipc2011-deterministic.

The evaluation relies on the metric defined for the IPC2011 and has been focused on the performance of repairing a plan[7]. In particular, we measured the time and the quality score (i.e. the resulting plan length[8]), and the coverage of the strategy (i.e. the percentage of solved problems). For comparison reasons, test cases ran also with a replanning from scratch, and with LPG-ADAPT ([18]).

Each computation is allotted with a maximum of 100 secs of cpu-time. Tests have been executed on Ubuntu 10.04 with an Intel Core Duo@2.53GHz cpu and 4 GB of Ram. The software has been implemented in Java 1.6 and the (re)planner used for the resolution of the arising planning tasks has been *Metric-FF* ([1]).

	Time Score				Quality Score				Coverage			
	GR	HKR	REP	LPGA	GR	HKR	REP	LPGA	GR	HKR	REP	LPGA
ZT	**55.0**	27.0	25.9	5.6	53.4	53.4	**54.2**	26.2	**100**	**100**	**100**	70.9
HardZT	**47.0**	24.3	17.7	7.4	46.1	**48.9**	47.4	21.3	85.5	**90.9**	87.3	61.8
Rover	**46.0**	4.7	9.9	22.7	42.2	**41.3**	26.1	46.3	89.1	83.6	49.1	**100**
Total	**148.0**	56.0	53.6	35.7	141.7	**143.6**	127.7	93.8	**91.5**	**91.5**	78.8	77.6

Table 1. Time Score, Quality Score and Coverage, according to the IPC 2011 metrics, for the three strategies over all the problems and domains.

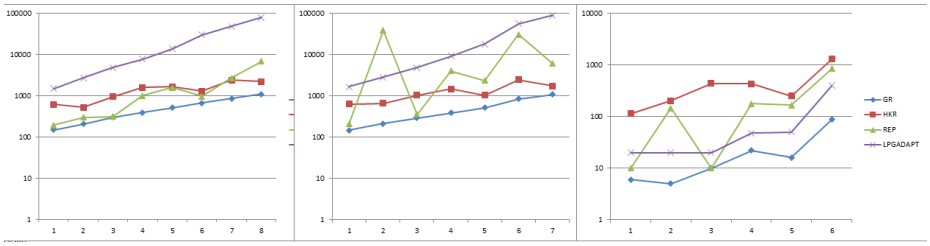

Fig. 1. The y-axis shows the Cpu-Time (in msec) for all the problems (x-axis) in the tested domains: ZenoTravel (left), Hard ZenoTravel (center), Rover (right)

Results are summarized by Table 1 and Figure 1; GR, HKR, REP and LPGA stand for, respectively, the greedy repair, the heuristic kernel repair, the replanning from scratch strategy and LPG-ADAPT. The table reports the scores obtained, while Figure 1 analyzes the scalability of the three approaches w.r.t. the increase of the difficulty of the problems[9].

Concerning the time score, GR outperformed the other strategies in all the tested domains (Table 1). The difference is large and it is also evident analyzing the gain between the curves reported in Figure 1. As matter of facts, GR

[7] It is worth noting that the kernel construction can be done in a preprocessing phase so that it can be seen as a form of off-line reasoning which hence does not influence the on-line performances.

[8] For the repair such a length is given by the patch plan and the plan suffix to execute.

[9] This scalability measures only those cases which have been successfully solved by all the strategies tested.

remains stably under the 1.5 secs of computational time for all the tested cases. Surprisingly, even for the quality score and the coverage, both the kernel based mechanisms outperformed the other strategies. This has been due to the many timeout situations encountered by running REP and LPGA. HKR is the one with the highest quality score but it is no competitive with GR time score. From our experiments we noticed that a large part of the HKR solving-time has been spent to compute the heuristic function (50%, 44% and 71% respectively for the ZenoTravel, Hard ZenoTravel and Rover). The bottleneck has been probably a not well optimized Java code (differently from others well known C++ implementations of the numeric planning graph heuristic, see for instance [1]); hence, we expect that the performance of HKR could be greatly increased.

6 Conclusions

The paper has proposed the notion of *numeric kernel*, which generalizes the STRIPS kernel for dealing with actions involving changes for continuous variables, i.e. numeric fluents. Concretely a *numeric kernel* is the weakest set of conditions that must be satisfied to make a plan of actions applicable. The paper formally reported a concrete regression mechanism for the construction of such a kernel, and interesting properties arising from its formulation.

Besides this theoretical contribution, the paper provided two effective strategies, which directly apply the kernel facilities for a problem of repair in presence of continuous and consumable resources. As a difference w.r.t. the most of the works appeared in literature ([19], [20], [18], [13]) in the context of on-line planning, the repair mechanisms implemented in this paper can be applied not only for the propositional setting, but also for the numeric one. Moreover, since these strategies do not make any assumption on the planner/heuristic used, they could be easily adapted with existing replanning/plan-adaption tools ([18]).

The performance of the repair strategies have been empirically evaluated on three domains defined by the planning community. Results showed that, in facing unexpected resources consumption, the proposed repair mechanisms are quite efficient, and produce good quality solutions, outperforming in the most of the cases both a replanning mechanism, and the LPG-ADAPT system ([18]). Such results, of course, need further confirmations on a larger set of domains; therefore, as an immediate future work, we are working on extending the experimental phase to assess the generality of the proposed strategies, possibly in combination with others plan-adaption tools.

From a methodological point of view, we are also studying how and when the *numeric kernel* notion can be employed in the context of case based planning with numeric fluents ([21]). Intuitively, we believe in fact that the notion may provide a quite good guidance in the selection/exploitation of past plans, solutions of (hopefully) similar problems.

References

1. Hoffmann, J.: The metric-ff planning system: Translating "ignoring delete lists" to numeric state variables. Journal of Artificial Intelligence Research **20** (2003)

291–341
2. Gerevini, A., Saetti, I., Serina, A.: An approach to efficient planning with numerical fluents and multi-criteria plan quality. Artificial Intelligence **172**(8-9) (2008) 899–944
3. Coles, A.J., Coles, A., Fox, M., Long, D.: Colin: Planning with continuous linear numeric change. Journal of Artificial Intelligence Research **44** (2012) 1–96
4. Fox, M., Long, D.: Pddl2.1: An extension to pddl for expressing temporal planning domains. Journal of Artificial Intelligence Research **20** (2003) 61–124
5. Coles, A.J., Coles, A.I., Fox, M., Long, D.: Forward-chaining partial-order planning. In: Proc. of International Conference on Automated Planning and Scheduling (ICAPS-10). (2010)
6. Conrad, P.R., Williams, B.C.: Drake: An efficient executive for temporal plans with choice. Journal of Artificial Intelligence Research **42** (2011) 607–659
7. Dechter, R., Meiri, I., Pearl, J.: Temporal constraint networks. Artificial Intelligence **49**(1-3) (1991) 61–95
8. Kvarnström, J., Heintz, F., Doherty, P.: A temporal logic-based planning and execution monitoring system. In: Proc. of International Conference on Automated Planning and Scheduling (ICAPS-08). (2008) 198–205
9. Policella, N., Cesta, A., Oddi, A., Smith, S.: Solve-and-robustify. Journal of Scheduling **12** (2009) 299–314
10. Stergiou, K., Koubarakis, M.: Backtracking algorithms for disjunctions of temporal constraints. Artificial Intelligence **120**(1) (2000) 81–117
11. Fikes, R., Hart, P.E., Nilsson, N.J.: Learning and executing generalized robot plans. Artificial Intelligence **3**(1-3) (1972) 251–288
12. Fritz, C., McIlraith, S.A.: Monitoring plan optimality during execution. In: Proc. of International Conference on Automated Planning and Scheduling (ICAPS-07). (2007) 144–151
13. Garrido, A., C., G., Onaindia, E.: Anytime plan-adaptation for continuous planning. In: Proc. of P&S Special Interest Group Workshop (PLANSIG-10). (2010)
14. Brenner, M., Nebel, B.: Continual planning and acting in dynamic multiagent environments. Journal of Autonomous Agents and Multiagent Systems **19**(3) (2009) 297–331
15. Chen, Y., Wah, B.W., wei Hsu, C.: Temporal planning using subgoal partitioning and resolution in sgplan. Journal of Artificial Intelligence Research **26** (2006) 369
16. Scala, E.: Reconfiguration and Replanning for robust Execution of Plans Involving Continous and Consumable Resources. Phd thesis in computer science, Department of Computer Science - Universita' di Torino (2013)
17. Nebel, B., Koehler, J.: Plan reuse versus plan generation: A theoretical and empirical analysis. Artificial Intelligence **76**(1-2) (1995) 427–454
18. Gerevini, A., Saetti, A., Serina, I.: Case-based planning for problems with real-valued fluents: Kernel functions for effective plan retrieval. In: Proc. of European Conference on AI (ECAI-12). (2012) 348–353
19. van der Krogt R., de Weerdt M.: Plan repair as an extension of planning. In: Proc. of International Conference on Automated Planning and Scheduling (ICAPS-05). (2005) 161–170
20. Fox, M., Gerevini, A., Long, D., Serina, I.: Plan stability: Replanning versus plan repair. In: Proc. of International Conference on Automated Planning and Scheduling (ICAPS-06). (2006) 212–221
21. Gerevini, A., Roubícková, A., Saetti, A., Serina, I.: On the plan-library maintenance problem in a case-based planner. In: International Conference on Case Based Reasoning (ICCBR-13). (2013) 119–133