Integrated Computer-Aided Engineering 22 (2015) 343–360
DOI 10.3233/ICA-150493
IOS Press

Reactive execution for solving plan failures in planning control applications

Cesar Guzman[a], Pablo Castejon[a], Eva Onaindia[a,*] and Jeremy Frank[b]
[a]*Universitat Politècnica de València, Valencia, Spain*
[b]*NASA Ames Research Center, Moffet Field, CA, USA*

Abstract. We present a novel reactive execution model for planning control applications which repairs plan failures at runtime. Our proposal is a domain-independent regression planning model which provides good-quality responses in a timely fashion. The use of a regressed model allows us to work exclusively with the sufficient and necessary information to deal with the plan failure. The model performs a time-bounded process that continuously operate on the plan to recover from incoming failures. This process guarantees there always exists a plan repair for a plan failure at anytime. The model is tested on a simulation of a real-world planetary space mission and on a well-known vehicle routing problem.

Keywords: Reactive planning, dynamic execution, monitoring plan execution, reactive execution agent, unpredictable environment

1. Introduction

The application of Artificial Intelligence AI planning techniques is helping industries to improve their efficiency and performance in a great variety of applications: manufacturing and telecommunication networks [31,37], education [20], route planning [10,41], military and civilian coalition operations [36], space exploration [9], etc.

In general, even though much of the research on AI planning is aimed at generating domain-independent planning technology, the application of planning to industry gives rise to special-purpose systems, which are expensive to extend to other cases. On the other hand, there exist few systems that integrate automated planning and plan execution and this is, perhaps, one of the main causes of the relatively low deployment of automated planning applications [22]. While the primary focus of planning is on deliberative tools to calculate plans that achieve operation goals, the focus of execution is on developing control methods over relatively short time spans to ensure the plan actions are executed stably [3].

Most planning and execution (P&E) systems follow an integrated approach in which the execution monitoring system is integrated with the planner or vice versa. Systems like IxTeT-eXeC [29] or TPOPEXEC [46] work under a continual planning approach [6], interleaving planning and execution in a world under continual change. Another example can be found in [9], where the planner of a spacecraft continuously operates on the plan execution to repair failures. IDEA (Intelligent Distributed Execution Architecture) [1] is a real-time architecture that proposes a unified view of deliberation and execution where the planner is embedded within the executor; and T-REX [32](Teleo-Reactive EXecutive) is a deliberative P&E system for AUV control inspired from IDEA. This type of unified approaches allows only for a strict and controlled interleaving of P&E, making it difficult to have a general-purpose planner for different types of executor systems.

Some of the aforementioned P&E systems [29,46], deal with temporal plans and focus on a unified approach that accommodates flexible plan executions, but

*Corresponding author: Eva Onaindia, Department de Sistemas Informáticos y Computación, Universitat Politècnica de València, Spain. Tel.: +34 963 877 755; Fax: +34 963 877 359; E-mail: onaindia@dsic.upv.es.

ISSN 1069-2509/15/$35.00

they are not concerned with providing responses in a timely fashion. In contrast, the works in [1,32], besides generating plans over relatively long time periods, they also introduce a reactive planner to allow robust performance in dynamic environments. The term reactive planning has been approached from different perspectives [8]:

- Responding very quickly to changes in the environment through a reactive plan library that stores the best course of action to each possible contingency.
- Choosing the immediate next action on the basis of the current context; in this case, a deliberative procedure can be used to specify the next act.
- Using more complex constructs in order to handle execution failures or environmental changes.

The first approach, used by early P&E systems, implies storing a plan for each possible state of the world, an option which is not affordable in highly dynamic environments. Hierarchical control structures provide a mechanism to choose the immediate next action when a quick response is required in unpredictable environments [7]. This is the approach followed by the models that emphasize reactiveness, computing just one next action in every instant, based on the current context [32]. The use of more complex structures allow to consider more deliberative (long horizon) responses rather than short-term reactiveness but, in practice, none of these reactive frameworks have ever exploited the idea of providing quick deliberative responses. They react to a change or failure but they do not guarantee a response within a limited time.

A key aspect of reactive planning is that it is not only about providing quick responses to changes but also preserving the plan stability [18] and guarantee that the operation goals achieved by the plan are still reachable after the plan repair. In contrast to control applications that use condition monitoring for anticipating and reacting to faults [4,40], reactive planning is about repairing a fault when is detected while trying to maintain the executability of the rest of the plan.

The focus of this work is on the development of a reactive P&E system capable to provide fast deliberative responses to repair a failed action without explicitly representing contingency branches and eligible plans for each possible state of the world. Our system does not simply return the immediate next executable action but it operates over a *planning horizon* that is longer than the minimum latency interval starting at the current execution time. This idea was exposed, though never exploited, in IDEA [13], which in practice works with the minimal planning horizon in order to reduce the reactive planner's search space; that is, the shorter the planning horizon, the more reactivity, but also the less contextual information to repair the failure. The idea of planning horizon has been also exploited in non-reactive dynamic planning applications [47].

In this work, we propose a novel reactive P&E model that keeps track of the execution of a plan and repairs the incoming failures. The executor-repairing system incorporates a reactive planning procedure which is exclusively used for plan repair and it is independent of the deliberative planner that computes the solution plan for the problem. Unlike the integrated P&E approaches mentioned before, ours is a highly modular and reconfigurable P&E architecture. The reactive planner is specialized in small plan repairs that must be accomplished promptly. Additionally, it precomputes an initial search space which encodes solution plans for potential failures in a fragment of the plan named *plan window* (we use this term equivalently to the concept of *planning horizon* in IDEA [1]). The construction of the search space is a time-bounded process subject to the agent's execution latency and the number of execution cycles in the plan window, whose objective is to have a solution plan available when a failure arises. Once the search space is built, the agent proceeds with the plan execution, and simultaneously the reactive planner computes the search space of the next plan window. The reactive model is thus capable to deduce strict limits to the length of the plan window and compute the largest search space within the time limit. Then, if a failure occurs, the corresponding search space is used to find a recovery plan. This gives our reactive model an *anytime-like* behaviour.

Our model contributes with several novelties: (a) it is a domain-independent P&E model that can be exploited in any application context; (b) it is independent of the deliberative planner; (c) it trades off deliberative and reactive mechanisms to provide good-quality responses in a timely fashion; (d) it avoids dealing with unnecessary information from the world, handling specifically the information relevant to the failure; and (e) it performs a time-bounded deliberative process that permits to continuously operate on the plan to repair problems during execution.

This paper is organized as follows. Section 2 presents some related work and Section 3 outlines the general P&E architecture where the reactive model is integrated. The two following sections present some formal concepts and introduce the reactive planning model to recover from plan failures. Section 6 presents

a motivation example on the Mars space mission. Section 7 presents the model evaluation, Section 8 discusses some limitation of the model and, finally, the last section concludes and outlines future research lines.

2. Related work

Classical planning refers generically to planning for state-transition systems that adopt a series of assumptions like that the system is deterministic and static, that actions are instantaneous transitions and that the planner is not concerned with any change that may occur in the system while it is planning [21]. In contrast, reactive planning operates in a timely fashion with highly dynamic, non-deterministic and unpredictable environments, assuming uncertainty in the world and the existence of multiple outcomes due to action failures or exogenous events [34]. Our reactive planner is not a temporal planner but it is designed to return timely responses in highly dynamic environments.

Regarding non-deterministic planning, the study of finding the sequence of actions or events that explain the current observed state of the world is called diagnosis. Most of the research on diagnosis put the emphasis in malfunctioning components and obtaining a plan to recover from the component failure rather than monitoring a plan execution [5,43]. Particularly, the work in [43] provides a formal characterization of diagnosis and its relation to planning and, in [5], authors propose an evolutionary strategy that successfully diagnoses several types of component failures, such as the separation of a body part or the complete failure of a sensor or motor.

Planning in non-deterministic environments has also been addressed from a probabilistic perspective, representing probabilities over the expected action outcomes or belief state space. Planning based on Markov Decision Processes is a well-known approach to deal with non-determinism when an accurate probability distribution on each state transition is available [27,33]. In highly dynamic environments where exogenous events frequently occur, it is not possible to have a model of the uncertainty in the world. Likewise, a Finite State Machine (FSM) can also be used to implement a reactive behaviour [39] but FSM requires an explicit modeling of a finite set of plans (states and transitions) and it is particularly aimed at choosing only the immediate next action. When a plan is to be executed in an unpredictable environment, it is impractical to have all potential repair plans explicitly represented and the emphasis is not only on short-term reactivity but also on preserving the achievability of the operation goals. For this reason, a time-bounded reactive planner that calculates promptly recovery plans over a planning horizon is the more suitable solution.

3. An architecture for planning and execution

Our work takes place in the context of PELEA [24], a single-agent architecture in which an agent is endowed with capabilities for generating a plan, executing, plan monitoring and, optionally, learning. Our ultimate goal of extending this model to a multi-agent context is discussed in Section 8. Before addressing this issue, our purpose is to have an agent equipped with a reactive execution mechanism that enables the agent to repair a plan at runtime, thus avoiding the need to resort to the deliberative planner each time a failure occurs. In the following, we will refer to the concept of agent, specifically to execution agent, as an autonomous entity capable of performing reasoning and communicating with other entities of the system like the deliberative planner, which offers a planning service.

In our approach, a planning service provides execution agents with independent plans to be executed. An agent, which is an extension of a PELEA[1] agent [24], executes and monitors one action at a time and calls its repairing mechanism for a recovery plan whenever a failure arises. In case the agent is unable to solve the failure by its own, it will request the planning service a new plan.

The focus of this paper is on the repairing mechanism of the execution agent. The planning module embedded into the execution agent is a reactive planner, which is used to recover from failures at runtime. The components of an execution agent (see Fig. 1) are:

- *Execution module* (EX). The EX is initialized with a planning task, which current state is read from the environment through the sensors. The EX is responsible of reading and communicating the current state to the rest of modules as well as executing the actions of the plan in the environment.
- *Monitoring module* (MO). The main task of the MO is to verify that the actions are executable in

[1] A more detailed description may be found at http://servergrps.dsic.upv.es/pelea/.

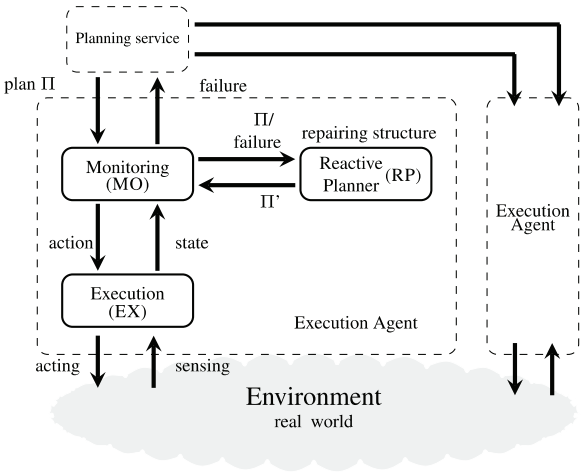

Fig. 1. Flow of the reactive execution model.

the current state before sending them to the EX. When the EX reports the MO the state resulting from the execution of some action of the plan, the MO checks whether the next action of the plan is executable in the resulting state. This process is called *plan monitoring*, verifying whether the values of the variables of the received state match the expected values or not. Otherwise, the MO will determine the existence of a plan failure.

– *Reactive Planner module* (RP). The RP is used when a plan failure is detected by the MO and a recovery is required. The RP uses a precomputed search space, called *repairing structure*, to promptly find a plan that brings the current state to one from which the plan execution can be resumed (see Section 5). In case the RP is not able to find a plan with its repairing structure, the MO requests the planning service a new plan.

The EX, MO, and RP modules of an execution agent operate the *Reactive Execution Model*. The control flow of the reactive execution model is shown in Fig. 1. An action plan Π for solving a planning task is calculated by the planning service. Π consists of a series of actions to be executed at given time steps, each of which makes a deterministic change to the current world state. The elapsed time from one time step to the next one defines an execution cycle; i.e., the *monitor/acting/sensing* cycle of an action execution. The model follows several execution cycles, performing the scheduled action in each cycle until the plan execution is completed.

Initially, the MO receives the plan Π from the planning service and, before sending Π to execution, it performs two operations:

1. It sends Π to the RP, which creates a *repairing structure* for a fragment of Π of length l called *plan window*, where l is the number of actions or execution cycles in the plan window. The repairing structure associated to the plan window contains information, in the form of alternative plans, to repair a failure that affects any of the l actions included in the plan window.
2. When the time of the RP expires, and a repairing structure has been calculated for a particular plan window, the MO monitors the variables of the first action of the window. If the sensed values of the action variables match the required values for the action to be executed, the MO sends the scheduled action to the EX for its execution (see Fig. 1). Otherwise, a failure is detected and the MO calls the RP, which will make use of the repairing structure to fix the failing action. Notice that a failure that occurs in the first action of a window is due to an exogenous event (e.g. other agents change the world state) and not due to an erroneous execution of the preceding action in the plan.

The MO receives the result of the sensing task from the EX after executing the action, it updates the plan window accordingly by eliminating the already executed action and proceeds with the next action of the plan window. For instance, assuming a plan of five actions and a repairing structure for a plan window of $l = 3$ ($[a_1, a_2, a_3]$), the plan window will be updated to $[a_2, a_3]$ after successfully executing a_1, and the RP will use the same repairing structure to fix a potential failure in the remaining actions of the plan window, that is, a_2 or a_3. Subsequently, the RP will create a new repairing structure, for example, for the plan window $[a_4, a_5]$. The flow goes on as long as no plan failures are encountered. In case that a non-executable action is found, the MO reports the failure to the RP. Then, the RP uses the repairing structure associated to the plan window of the non-executable action and obtains a new plan Π' that solves the failure and replaces the old plan Π, attaining likewise the goals of the planning task.

The RP is constantly working while the EX is executing the actions of the plan. Hence, besides having a repairing structure ready to attend a failure in an action of the current plan window, the RP is also generating the subsequent structure for the next plan window. Typically, the time for the RP to compute the repairing structure of the subsequent plan window is the time that the EX will take to execute the actions included in the current window. Therefore, the

more actions in the current plan window, the more time the RP will have to create the next repairing structure and, consequently, the longer the window associated to this repairing structure. This working scheme gives our model an anytime behaviour, thus guaranteeing the RP can be interrupted at anytime and will always have a repairing structure available to attend an immediate plan failure.

On the other hand, some similarities between our model and the life cycle of a scientific workflow can be found. Following [23], we can do this analogy: the modeling phase is equivalent to the planning task modeling; the deployment phase amounts to the plan Π calculated by the planning service; and the execution and monitoring phase would be the same in our model. Unlike scientific workflow, our model does not include an analysis phase; however, successively repetitions of the deployment (repair plan) and execution phases happen when a plan failure arises.

4. Formal model

In this section, we formalize the concept of planning task, partial state and a solution plan for a task as a sequence of partial states [21]. Our planning formalism is based on a multi-valued state-variable representation where each variable is assigned a value from a multiple value domain (finite domain of a variable). For modeling planning problems, we used PDDL3.1,[2] the most recent version of the Planning Domain Definition Language [17] (PDDL).

Definition 1. Planning task A planning task is given by the 4-tuple $\mathcal{P} = \langle \mathcal{V}, \mathcal{I}, \mathcal{G}, \mathcal{A} \rangle$:

- $\mathcal{V}$ is a finite set of **state variables**, each associated to a finite domain, $\mathcal{D}_v$, of mutually exclusive values that refer to objects in the world. $v \in \mathcal{V}$ maps a tuple of objects to an object p of the planning task, which represents the value of v. For example, in a planetary Mars rovers domain,[3] a rover (B) can be placed at any of the waypoints $\mathtt{w}_1$, $\mathtt{w}_2$ or $\mathtt{w}_3$. Hence, the variable loc-B represents the location of rover B, and $\mathcal{D}_{\texttt{loc-B}} = \{\mathtt{w}_1, \mathtt{w}_2, \mathtt{w}_3\}$.
  A variable assignment or *fluent* is a function f on a variable v such that $f(v) \in \mathcal{D}_v$, wherever $f(v)$ is defined. A fluent is represented as a tuple $\langle v, p \rangle$, meaning that the variable v takes the value p.
  A total variable assignment or *state* applies the function f to all variables in $\mathcal{V}$. A state is always interpreted as a *world state*. A partial variable assignment or *partial state* over $\mathcal{V}$ applies the function f to some subset of $\mathcal{V}$.
- $\mathcal{I}$ is a state that represents the *initial state* of the planning task.
- $\mathcal{G}$ is a partial state over $\mathcal{V}$ called the problem *goal state*.
- $\mathcal{A}$ is a finite set of *actions* over $\mathcal{V}$. An action a is defined as a partial variable assignment pair $a = \langle pre, eff \rangle$ over $\mathcal{V}$ called *preconditions* and *effects*, respectively. If an action is executable in a state, i.e. its preconditions hold in such a state, the values of the state variables (fluents) change as specified in the effects.

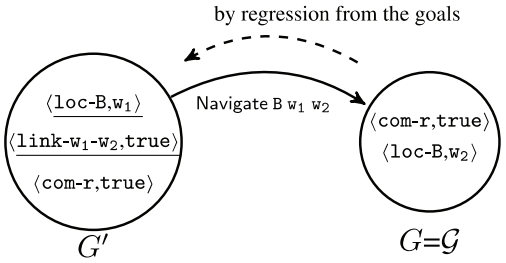

Fig. 2. Plan as a sequence of partial states. Variables are loc-B: location of rover B; link-$\mathtt{w}_1$-$\mathtt{w}_2$: map to travel from $\mathtt{w}_1$ to $\mathtt{w}_2$; com-r: communication of the results of analyzing the rock r. Underlined variables are the preconditions of the action Navigate.

An action plan, Π_A, that solves a planning task $\mathcal{P}$ is a sequence of actions $\Pi_A = \langle a_1, \ldots, a_n \rangle$ that applied in the initial state $\mathcal{I}$ satisfies the goal state $\mathcal{G}$. An action $a_i \in \Pi_A$ is executable in a world state S if the fluents contained in S satisfy the preconditions of a_i; i.e. $pre(a_i) \subseteq S$. The result of executing a_i in a state S is a new state S' that contains the fluents of S which are not modified by *eff*(a_i) plus the fluents as specified in *eff*(a_i). Then, executing Π_A in the initial state $\mathcal{I}$ results in a sequence of states $\langle S_1, \ldots, S_n \rangle$ such that S_1 is the result of applying a_1 in $\mathcal{I}$, S_2 is the result of applying a_2 in $S_1, \ldots$, and S_n is the result of applying a_n in S_{n-1}. A plan Π_A is a solution plan iff $\mathcal{G} \subseteq S_n$ [21].

A plan can also be viewed from the point of view of the world conditions (fluents) that are necessary for the plan to be executed. That is, instead of viewing a plan as the result of its execution, we can view a plan as the *necessary conditions* for its execution. Thus, a plan can also be defined as a sequence of partial states, rather than world states, containing the *minimal set of*

[2] PDDL syntax definition introduced in 2008 by M. Helmert (http://ipc.informatik.uni-freiburg.de/PddlExtension/).

[3] Our PDDL specification of this domain can be found at http://servergrps.dsic.upv.es/planinteraction/resources/.

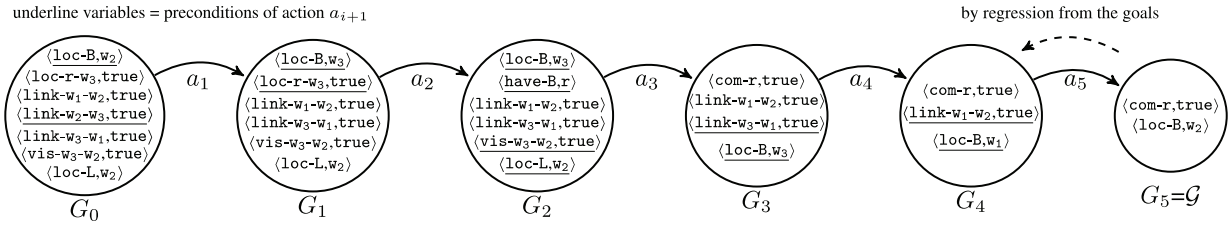

Fig. 3. Plan as a sequence of partial states for the plan Π_A. Variables are loc-B: location of rover B; loc-L: location of lander L; loc-r-w_3: location of the rock r; have-B: indicates if B has the rock r; link-w_i-w_j: map to travel from w_i to w_j; vis-w_3-w_2: location w_2 is visible from w_3. com-r: results of analyzing the rock r are communicated.

fluents that must hold in the world for the plan to be executable in such a world state.

In the example depicted in Fig. 2, the partial state G is the goal state $\mathcal{G}$ of a planning task $\mathcal{P}$, and it contains two fluents. Let's consider the last action of a plan is (Navigate B w_1 w_2), which achieves the effect $\langle$loc-B, $w_2\rangle$. Then, the necessary fluents to be able to execute the action and achieve the fluents in $\mathcal{G}$ are represented in state G'. We can observe that G' does not only contain the fluents that match the preconditions of the action Navigate (i.e., $\langle$loc-B, $w_1\rangle$ and $\langle$link-w_1-w_2, true$\rangle$, which represent that the location of rover B must be the waypoint w_1 and a link between w_1 and w_2 must exist, respectively) but also the fluent $\langle$com-r, true$\rangle$. This is because this fluent (communicating the results of analyzing the rocks r) is a goal of $\mathcal{G}$ that is not achieved by the effects of the action Navigate. Thereby, $\langle$com-r, true$\rangle$ is achieved earlier in the plan and it must hold in G' in order to guarantee that it is satisfied in $\mathcal{G}$.

The state G' in Fig. 2 is called a *regressed partial state* because it is calculated by regressing the goals in $\mathcal{G}$ through the action Navigate. Likewise, the same regression can be applied to the rest of actions of a given plan Π_A. Let a_i be an action and G a goal state such that $P = pre(a_i)$, $E = eff(a_i)$ and $E \subseteq G$. The *partial state* G' in which a_i can be applied is calculated by the *regressed transition function* $\Gamma(G, a_i)$, defined as:

$$G' := \Gamma(G, a_i) := G \setminus E \cup P \quad (1)$$

G' is a partial state that represents the minimal set of fluents that must hold in the world state in order to achieve G by means of the execution of a_i. Notice that G' includes P, the preconditions of a_i, plus the fluents which are in G but are not produced by E ($G \setminus E$); i.e., the fluents that are achieved before G' in the plan and must keep their values until G.

The regressed partial state approach was first used by PLANEX [11] to supervise the execution of a sequence of actions. Plans are represented by means of a triangle table (this structure provides support for plan monitoring) and the problem goals are regressed from the last column of the table, including action preconditions, through the remaining actions of the plan. Roughly, the regression of a fluent over an action (through the regressed transition function Γ) is a sufficient and necessary condition for the satisfaction of the fluent following the execution of the action. The work in [19] formalizes this concept in the situation calculus language whereas we apply the same formalization in PDDL.

	a_1	(Navigate B w_2 w_3)
	a_2	(Analyze r B w_3)
Π_A	a_3	(Communicate r B L w_3 w_2)
	a_4	(Navigate B w_3 w_1)
	a_5	(Navigate B w_1 w_2)

Fig. 4. Plan Π_A for a planetary Mars rover domain.

Definition 2. Solution plan as a sequence of partial states Given a solution plan $\Pi_A = \langle a_1, \ldots, a_n\rangle$ for a planning task $\mathcal{P}$, the *regressed plan* for Π_A is defined as a chronologically ordered sequence of partial states $\langle G_0, G_1, \ldots G_n\rangle$, where:

$G_n := \mathcal{G}$
$G_0 \subseteq \mathcal{I}$
$G_{i-1} := \Gamma(G_i, a_i)$

A regressed plan is denoted by $\Pi_{G_0-G_n}$, where G_0 is the initial partial state and G_n is the final partial state of the plan Π_A. Definition 2 specifies the *relevant* fluents at each time step for the successful execution of Π_A, where each $a_i \in \Pi_A$ is the relevant action for achieving G_i from G_{i-1}. Hence, a regressed plan $\Pi_{G_0-G_n}$ derived from Π_A denotes the fluents that must hold in the environment at each time step to successfully execute the actions in Π_A. In other words, this definition allows us to discern between the fluents that are relevant for the execution of a plan and those ones that are not. It exploits the idea of annotating plans

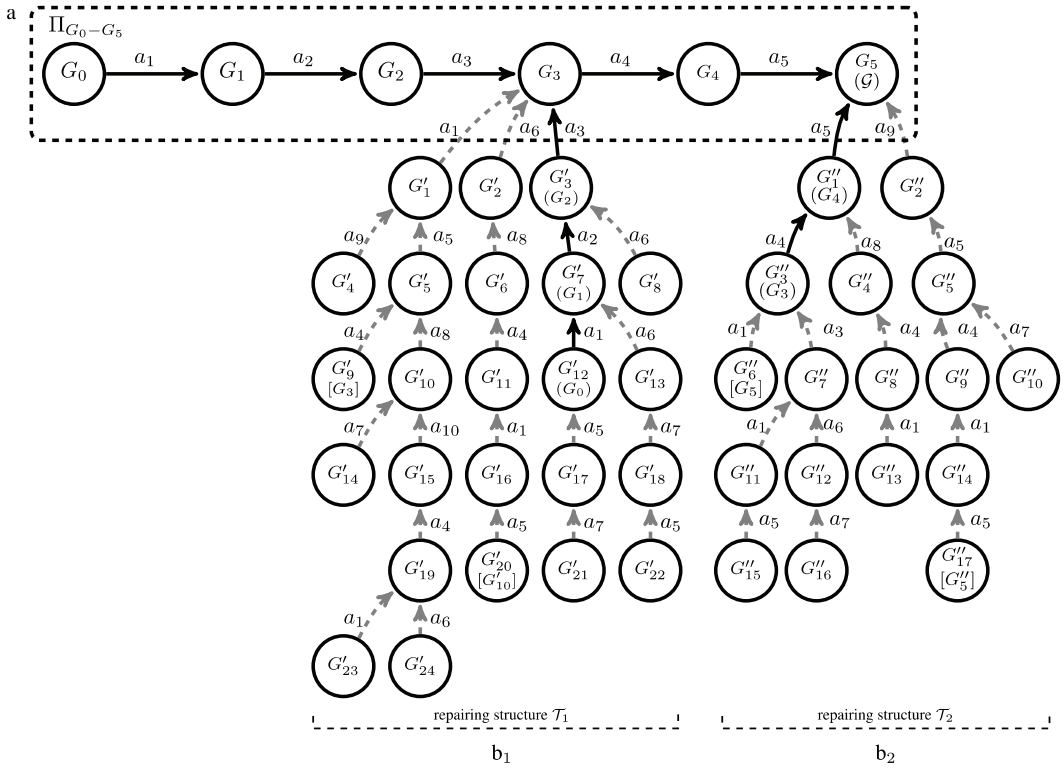

Fig. 5. Repairing structures for a rover B in a Planetary Mars Domain. a: regressed plan $\Pi_{G_0-G_5}$ for $\langle a_1, a_2, a_3, a_4, a_5\rangle$, b_1: the repairing structure of the plan window [a_1,a_2,a_3], and b_2: the repairing structure of the plan window [a_4,a_5].

with conditions that can be checked at execution time to confirm the validity of a plan [11]. That is, if the fluents of a partial state G_i hold in a world state S ($G_i \subseteq S$) then the actions comprised in the plan fragment $\Pi_{G_i-G_n}$ are executable in S, thus guaranteeing the goals of the planning task are achieved.

Figure 3 shows the regressed plan $\Pi_{G_0-G_5}$ derived from the solution plan shown in Fig. 4, and calculated through the successive application of the regressed transition function Γ. The plan in Fig. 4 shows the actions for a rover to gather a rock sample, communicate the results of analyzing the rock and navigate back to the initial position. Action a_2, for instance, creates the fluent $\langle$`have-B,r`$\rangle$ in G_2; and the partial state G_1 is the result from applying $\Gamma(G_2, a_2)$, which includes $pre(a_2)$ (the fluents which are underlined in node G_1 of Fig. 3) plus the fluents that are in G_2 but are not produced by *eff*(a_2). Therefore, if the sensor reading returns a world state in which all of the fluents in G_1 hold, then action a_2 is executable in such a world state; if the fluents in G_2 occur in the subsequent world state then a_3 is executable and so on. The last partial state, G_5, comprises $\mathcal{G}$, the goals of the planning task. In terms of plan monitoring, G_5 represents the fluents that satisfy the preconditions of a fictitious final action, a_f, where $pre(a_f) = \mathcal{G}$ and *eff*$(a_f) = \emptyset$, i.e. $\mathcal{G}$ is monitored by checking the preconditions of a_f.

As a final remark, we note that the reactive execution model is defined at the same granularity level than the planning model, and both use PDDL as the specification language. This eases the communication between the planning service and execution agents and avoids the overhead of translating a high-level planning specification into a low-level description as it happens in other models [14].

5. Reactive execution model

The key concept of our reactive execution model is the *repairing structure*. Generally speaking, given a solution plan Π_A, a repairing structure $\mathcal{T}$ is a partial-state search tree that encodes recovery plans for a plan window of Π_A. Since nodes in $\mathcal{T}$ are partial states, the reactive execution model only handles the minimal data set that is necessary to carry out a repairing task.

Figure 5 shows two repairing structures, $\mathcal{T}_1$ and $\mathcal{T}_2$, for the regressed plan in Fig. 3 (for simplicity, we do

not show all the partial states that would be generated). $\mathcal{T}_1$ is the search tree associated to the plan window $[a_1, a_2, a_3]$, or, equivalently, to the regressed subplan $\Pi_{G_0-G_3}$. The length of the window is three ($l = 3$), because it comprises three actions, and the partial state G_3 is called the *root node* (G_r) of $\mathcal{T}_1$. A path in $\mathcal{T}_1$ represents a (regressed) plan to reach the root node G_3. Specifically, a path in a repairing structure is interpreted as a recovery plan that leads the current world state to another state from which the execution of the plan Π_A can be resumed. All recovery plans in $\mathcal{T}_1$ have one thing in common: they eventually guide the execution of the plan towards G_3, the root node of $\mathcal{T}_1$. For instance, suppose that S is the set of fluents that represents the state of the world state such that $G'_{16} \subseteq S$ (see Fig. 5 b_1). The application of the plan $\Pi' = \langle a_1, a_4, a_8, a_6 \rangle$ to S will reach the partial state G_3, from which the rest of the plan Π_A, $\langle a_4, a_5 \rangle$, can be executed.

The tree $\mathcal{T}_2$ (see Fig. 5 b_2) is associated to the plan window $[a_4, a_5]$, or to the regressed plan $\Pi_{G_3-G_5}$. The number of repairing structures necessary to keep track of the execution of a plan Π_A depends on the time limit to create the search trees, which, in turn, delimits the size of the tree. Two parameters determine the size of a search space $\mathcal{T}$, the length of the plan window associated to $\mathcal{T}$ (l), and the depth of the tree (d). In general, the larger the value of l, the more alternatives to find a recovery plan; and the deeper the tree, the longer the recovery plans comprised in $\mathcal{T}$. The minimum value of d must be $l + 1$ in order to ensure that the tree comprises at least one action that repairs the first action of the plan window associated to $\mathcal{T}$. On the other hand, the maximum value of d is determined by the available time to build $\mathcal{T}$. Particularly, in $\mathcal{T}_1$, $d = 6$, which results from $l = 3$ and the time limit to build $\mathcal{T}_1$ (Section 5.3 explains in detail how to estimate the maximum size of a repairing structure).

In the following, we explain (1) the process to build a repairing structure, (2) how to find a plan in a search tree to repair a failure and (3) the analysis to estimate the size of the search tree.

5.1. Building a repairing structure $\mathcal{T}$

The construction of the repairing structure $\mathcal{T}$ starts after estimating the size of $\mathcal{T}$; i.e., when the values of l and d are known. The generation process, shown in Algorithm 1, consists in expanding $\mathcal{T}$ from the root node G_r via the application of the regressed transition function $\Gamma(G, a)$ following Eq. (1) (line 6 of Algorithm 1). The algorithm is a classical backward construction of a planning search space [21], where a node G is expanded until $depth(G) = d$; i.e., G reaches the maximum depth tree (line 4), or G is superseded by another node that exists in the tree (lines 7 to 13 define a mechanism for the control of repeated states which is detailed below).

Input: G_r, d
Output: $\mathcal{T}$
1: $\mathcal{Q} \leftarrow \{G_r\}, \mathcal{T} \leftarrow \{G_r\}$
2: **while** $\mathcal{Q} \neq \emptyset$ **do**
3: $G \leftarrow$ remove first node from $\mathcal{Q}$
4: **if** $depth(G) < d$ **then**
5: **for all** $\{a \mid a \in \mathcal{A}$ is a *relevant action* to $G\}$ **do**
6: $G' \leftarrow \Gamma(G, a)$
7: **if** $G' \notin \mathcal{T}$ **then**
8: **if** $\exists\, G'' \in \mathcal{T} \mid G'' \subset G'$ **then**
9: mark G' as *superset* of G''
10: **else**
11: $\mathcal{Q} \leftarrow \mathcal{Q} \cup G'$
12: set transition (labeled a) from G' to G
13: $\mathcal{T} \leftarrow \mathcal{T} \cup G'$
14: **else**
15: $\mathcal{Q} \leftarrow \emptyset$
16: **return** $\mathcal{T}$

Algorithm 1: Generating the repairing structure $\mathcal{T}$.

The purpose of Algorithm 1 is to generate multiple regressed plans from G_r. Unlike the application of $\Gamma(G, a)$ in Definition 2, which departs from a given solution plan Π_A, such a plan does not exist when building a tree $\mathcal{T}$. Actually, the aim of Algorithm 1 is precisely to find the relevant actions for a node G (line 5), and eventually create a plan Π_A that links two particular partial states.

The operation of regressing a fluent f in a node G over an action a checks whether a is a relevant action to achieve f or not. An action a is relevant for f, and originates an arc (G', G) in $\mathcal{T}$, if it does not cause any conflict with the fluents in G and G'. The construction of $\mathcal{T}$ has then to check two consistency restrictions: (1) that $eff(a)$ does not conflict with the fluents in G, and (2) that $pre(a)$ does not conflict with the fluents in G'.

We define $\Phi(G', G)$ as the function that returns whether or not a conflict between two sets of fluents G' and G exists. $\Phi(G', G)$ holds if $\exists \langle v, p \rangle \in G$ and $\exists \langle v, p' \rangle \in G'$ and $p \neq p'$.

Definition 3. Relevant action Given a fluent $f \in G$, a is a relevant action for f if the following conditions hold:

1) $f \in \mathit{eff}(a)$ and

2) $\neg\Phi(\mathit{eff}(a), G)$ and
3) $G' = \Gamma(G, a) \wedge \neg\Phi(pre(a), G')$

The construction of $\mathcal{T}$ follows the application of Definition 3 for each fluent of a partial state G which has not reached $depth(G) = d$ (lines 4 and 5 of Algorithm 1), and the expansion continues until no new partial states are added to the tree. The search space $\mathcal{T}$ is actually a graph due to the existence of multiple paths that reach the same partial state from the root node during the construction of $\mathcal{T}$. Multiple paths are originated because of actions like (Communicate `rock B L` w_3 w_2) and (Communicate `soil B L` w_3 w_2), which can be executed in either order, or the existence of reversible actions like (Navigate `B` w_1 w_2) and (Navigate `B` w_2 w_1). Consequently, $\mathcal{T}$ may contain many redundant paths. A set of state variables induce a state space that has a size that is exponential in the set, and, for this reason, planning, as well as many search problems, suffer from a combinatorial explosion. Even though nodes in $\mathcal{T}$ are partial states that contain far less fluents than world states, the large size of the repairing structures are sometimes unaffordable for a reactive system. With the aim of reducing the size of $\mathcal{T}$, we only consider for expansion the fluents of G that are related to the *relevant variables*, that is, the variables involved in the preconditions of the actions of the plan window. Thus, given a plan window $[a_1, a_2, a_3]$, we approximate $\mathcal{T}$ by expanding only the fluents related to the *relevant variables* involved in the set $pre(a_1) \cup pre(a_2) \cup pre(a_3)$, which is actually the set of fluents that might need to be repaired. The time complexity of Algorithm 1 responds to the classical complexity of the generation of a tree, that is $\mathcal{O}(\hat{b}^d)$, where $\hat{b}$ is the estimated branching factor of $\mathcal{T}$ that is detailed in Section 5.3.

The generation process makes two nodes in $\mathcal{T}$ be connected by a unique simple path. Since we are interested in keeping only the shortest (optimal) paths, the construction of $\mathcal{T}$ prunes repeated states (line 7 in Algorithm 1) and avoids the expansion of *superset* nodes (lines 8 and 9). Let's assume that $\mathcal{T}$ contains a path from a node G to the root node G_r of $\mathcal{T}$. A node G' such that $G \subset G'$ is said to be a superset of node G. In this case:

- G stands for the minimal set of fluents that must hold in S in order to execute the actions of the path that reaches G_r.
- The best recovery plan from G is also the best path from G' because the RP returns the shortest plan to G_r.

All in all, a repairing structure encodes the optimal path between each pair of nodes for which a recovery plan can be found. Once the RP has created $\mathcal{T}$, it communicates the MO all the variables involved in $\mathcal{T}$.

5.2. Repairing a failure

When an action of the plan window associated to a repairing structure $\mathcal{T}$ fails, the RP finds a way to keep the plan going, either by reaching a partial state in $\mathcal{T}$ from which to execute the faulty action again or rather another state from which to execute a later action of the plan window.

Let $\mathcal{T}$ be a repairing structure of a regressed plan $\Pi_{G_0-G_r}$ associated to the plan window $[a_1, \ldots, a_r]$ of a plan Π_A. When an action in $[a_1, \ldots, a_r]$ fails, a repairing task defined as $\mathcal{R} = \langle S, G_t \rangle$ is activated, where S[4] is the set of fluents of the current world state and G_t is the target state we want to reach in $\mathcal{T}$. The node G_t varies depending on the failed action and the particular repairing task for such action. Since several recovery plans can be found to fix a faulty action, the RP will successively execute a repairing task until one of them is successful for fixing the action. This way, if the erroneous action is a_1, the RP will first try the repairing task $\mathcal{R} = \langle S, G_0 \rangle$; otherwise, it will try $\mathcal{R} = \langle S, G_1 \rangle$ and so on until $G_t = G_r$; if the failure occurs in a_2, the first attempt will be $\mathcal{R} = \langle S, G_1 \rangle$ and the last attempt will be for $G_t = G_r$; in the case that the failure affects a_r, only two repairing tasks can be realized, $\mathcal{R} = \langle S, G_{r-1} \rangle$ and $\mathcal{R} = \langle S, G_r \rangle$.

More formally, given $\mathcal{R} = \langle S, G_t \rangle$ for a faulty action a, the RP applies a modified breadth-first search from G_t until a node G_s that satisfies $G_s \subseteq S$ is found in $\mathcal{T}$. G_s is a consistent state with S, a state that comprises all the necessary fluents to execute in the current world state the plan formed with the actions from G_s to G_t. If $G_s \subseteq S$ is found, the recovery plan from G_s to G_t is concatenated with the plan from G_t to G_r (unless $G_t = G_r$), and with the plan from G_r to G_n, where G_n is the last state of the original plan Π_A which contains $\mathcal{G}$, the problem goal state. If $G_s \subseteq S$ is not found, the RP will execute the subsequent repairing task $\mathcal{R} = \langle S, G_{t+1} \rangle$ until one of them successfully retrieves a recovery plan or $\mathcal{R}$ is invoked with $G_t = G_r$ and a plan is not found. In this latter case, $\mathcal{T}$ does not comprise the necessary information to find a

[4] Technically speaking, the MO does not communicate the RP all of the fluents in S but only the values of the variables that appear in $\mathcal{T}$; these variables were sent by the RP to the MO after building $\mathcal{T}$.

recovery plan and the planner service is invoked, which performs a replanning task.

Let's see how the repairing task procedure applies to a particular example. Consider that a failure occurs in the action a_1, Navigate B $\mathtt{w}_2$ $\mathtt{w}_3$, in the subplan $\Pi_{G_0-G_3}$ of the repairing structure $\mathcal{T}_1$ of Fig. 5 b_1. Let's assume that the failure is due to a wrong location of rover B, which is not at $\mathtt{w}_2$ but at $\mathtt{w}_1$.

- The first repairing task is $\mathcal{R} = \langle S, G_0 \rangle$, where $G_0 \subseteq G'_{12}$. The only two nodes that are reachable from G'_{12} are G'_{17} and G'_{21}. If none of these two states match the current world state S, that is $G'_{17} \nsubseteq S$ and $G'_{21} \nsubseteq S$, then it means no plan repair actually exists to reach G_0 from S in $\mathcal{T}_1$, and, hence, there is no way to return rover B to $\mathtt{w}_2$ from $\mathtt{w}_1$.
- Assuming there is no solution to move rover B back to $\mathtt{w}_2$, the RP attempts the next repairing task $\mathcal{R} = \langle S, G_1 \rangle$, where $G_1 \subseteq G'_7$ in $\mathcal{T}_1$ is a partial state in which B analyzes the rock at location $\mathtt{w}_3$ and communicates the results to the lander. Hence, for every descendant node G_s of G'_7 (excluding G'_{12} and its descendent nodes which were already explored in the previous repairing task), the RP checks whether it holds $G_s \subseteq S$, in which case the RP will return the plan from G_s to G'_7 concatenated with the plan $\langle a_2, a_3 \rangle$ and $\langle a_4, a_5 \rangle$. Let's assume that $G'_{13} \subseteq S$ holds, in which case a path that reaches G'_7 from S actually exists. This path is formed of the action a_6, Navigate B $\mathtt{w}_1$ $\mathtt{w}_3$, that moves the rover B from $\mathtt{w}_1$ to $\mathtt{w}_3$, where the rover has to analyze the rock. Then, the RP returns the recovery plan $\langle a_6 \rangle$, concatenated with $\langle a_2, a_3 \rangle$ (analyze the rock and communicate the results, respectively) and concatenated with $\langle a_4, a_5 \rangle$ (the rest of actions in Π_A).

Two issues related to the repairing task are worth mentioning here. First, it is important to highlight that the choices to find a recovery plan increase when the target state is closer to the root node of the tree. This can be graphically observed in Fig. 6. The top figure shows a tree $\mathcal{T}$ of depth d associated to a plan window of length $l = 3$. If the repairing task is $\mathcal{R} = \langle S, G_0 \rangle$, actions a_1, a_2 and a_3 must be included in the recovery plan, and the path-finding algorithm restricts the search to m levels of the tree, the shadowed portion of the tree in the figure. If, however, our repairing task is $\mathcal{R} = \langle S, G_1 \rangle$ (bottom figure) then the recovery plan must only comprise a_2 and a_3, and, the choices to find a state that matches S increase as well as the recovery plans to reach G_1. In conclusion, when the target state

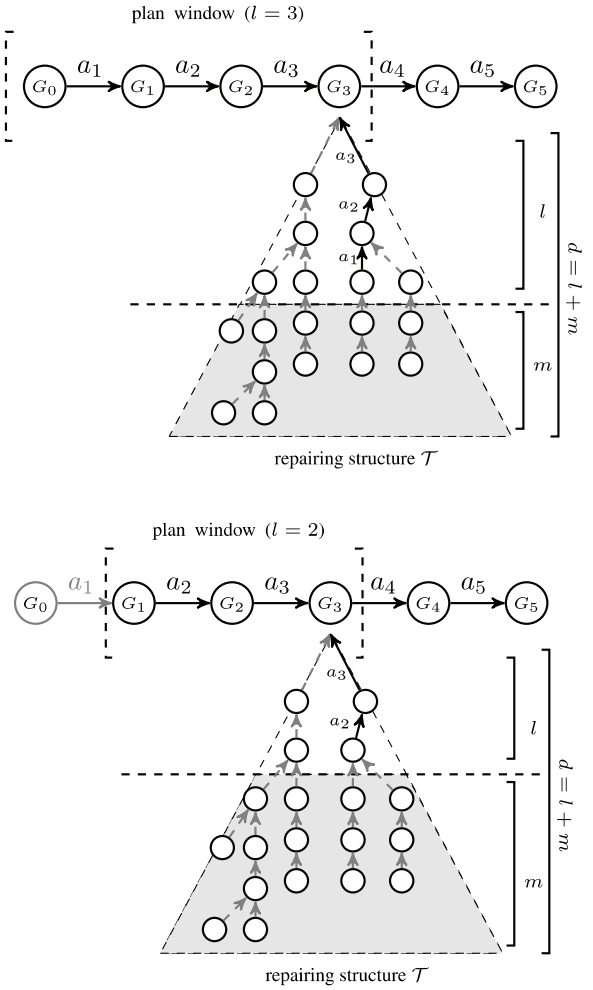

Fig. 6. Repairing structures abstract.

is one of the first states of the subplan, we find fewer alternatives of repairing but the recovery plan guarantees more stability with respect to the original plan. In contrast, if the target state is closer to the root node then there are more choices to find a recovery plan although the plan found might not keep any of the actions in the original plan. Clearly, the more flexibility, the less stability.

Secondly, it might happen that no recovery plan is found for a repairing task. Notice that the tree has a limited size in terms of l and d that is determined by the available time to build the tree, and the information included in the tree may not be sufficient to solve all the potential contingencies. The principle underpinning reactive systems is that of providing a prompt reply, and this requires to work with bounded data structures. Moreover, reactive responses are applied when slight deviations from the main course of actions occur.

A major fault that makes a variable take a value that is not considered in the repairing structure would likely need a more deliberative response.

5.3. Estimating the size of $\mathcal{T}$

In order to ensure reactivity, we need some guarantee that a repairing structure $\mathcal{T}$ is available when a failure arises and that the only operation that needs to be done is to find a recovery plan in $\mathcal{T}$. In this section, we explain the details of the time-bounded building of $\mathcal{T}$.

The time limit (t_s) to build some $\mathcal{T}$ is given by the agent's execution latency (the longest acceptable time for the agent to start the execution of an action and return the world state resulting from such execution), and the number of execution cycles (actions) in the plan window of the preceding $\mathcal{T}$. The agent's latency depends on whether the system that is being observed is simulated or real and the characteristics of the agent's domain. Some domains may require a relatively large execution latency (e.g., 10 s.), but others may have a much shorter latency (e.g., 10 ms.). In our experiments (see Section 7) we assume a latency of 1000 ms.

Estimating variables that have a long range of values, like the time to generate $\mathcal{T}$ ($t_{\mathcal{T}}$), with a multiple linear regression model may produce highly inaccurate predictions [26]. Therefore, our proposal is to estimate the branching factor of $\mathcal{T}$ (b) instead of directly estimating $t_{\mathcal{T}}$; the estimation of b is given by the Eq. (2a). Our estimation model relies on several parameters that affect the size of $\mathcal{T}$, namely, the depth d of $\mathcal{T}$ (x_1); the number of fluents in G_r (x_2); the number of *relevant variables* associated to $\mathcal{T}$ (x_3) as well as the sum of the domain cardinalities of these variables (x_4); the number of relevant actions that modify these variables without removing duplicates (x_5) as well as removing them (x_6); and the number of these variables that also appear in G_r (x_7). The values from x_2 to x_7 depend on the length (l) of the plan window, hence, our estimation depends mainly on the depth d and the length l of $\mathcal{T}$. We preserve the value of d in our estimation model to ensure that $\mathcal{T}$ will comprise at least one action to repair the first action of the plan window. In order to learn the weights w_i, we designed a series of experiments to generate random repairing structures for different problems from diverse planning benchmarks[5] and obtain the value of b of the generated trees. Since these trees are irregularly shaped, we approximated the value of b ($\hat{b}$) through Eq. (2b) (an approximation to the number of nodes N in a uniform tree). Finally, the time to apply the regressed transition function Γ to a G_i ($\bar{t_\Gamma}$) is computed as the average of the ratios between the time of generating a tree and the number of nodes of such a tree over the whole benchmark.

Table 1
Trace to calculate l and d values with $t_s = 1000$ ms

l, d	2, 3	2, 4	2, 5	3, 4	3, 5	4, 5
$\delta(l, d)$	255	637	~~1052~~	795	~~1230~~	~~1084~~
Selected				✓		

$$\hat{b} = w_0 + \sum_{i=1}^{7} w_i * x_i \tag{2a}$$

$$\tilde{N} = (\hat{b} + 0.34)^d \tag{2b}$$

$$\hat{t}_{\mathcal{T}} \equiv \delta(l, d) = \tilde{N} * \bar{t_\Gamma} \tag{2c}$$

$$\{l, d\} = \underset{\substack{l \in [2, x],\, d \in [l+1, y] \\ \delta(l, d) < t_s}}{\arg\max} \delta(l, d) \tag{2d}$$

Once the estimation model is trained, in the testing stage we use the Eq. (2d) to find the values of l and d that maximize $\hat{t}_{\mathcal{T}}$ within the time limit t_s. Through this maximization process we compute for every pair of l and d: (1) the associated value of $\hat{b}$, (2) the number of partial states $\tilde{N}$, and (3) whether or not the $\hat{t}_{\mathcal{T}}$, result of Eq. (2c), is lower than t_s. Table 1 shows a trace of this process. The RP starts with the combination $l = 2$ and $d = 3$ and the value of d is progressively increased by 1 until $\delta(l, d) > t_s$ (see $\delta(2, 5)$ in Table 1). Then, the value of l is increased by 1 and d resets to $l + 1 = 4$. Since $\delta(l, d)$ is an increasing function with the values of l and d, their upper boundaries, x and y, will be limited according to the value of t_s (in Table 1 these upper boundaries are 4 and 5, respectively). Finally, the RP selects the closest combination to t_s ($\delta(3, 4)$ in Table 1) and the remaining time $\epsilon = t_s - \hat{t}_{\mathcal{T}}$ is added to the t_s for the next structure.

6. Planetary Mars domain, a real-world motivation example

The Mars planning domain stems from a real-world problem that NASA is dealing with in some research projects on space exploration [30]. In this domain, a rover that works on the Martian surface may have different instruments (e.g. cameras, robotic arms, drills, atmospheric sensors) to analyze rocks or soils, to take

[5] Part of the benchmark suite and the training data set can be found at http://servergrps.dsic.upv.es/planinteraction/resources/.

images of the terrain, or perform atmospheric measurements over many location types on the Mars surface. Location types included soft sand, hard-packed soil, rough rock fields, and combinations of these all. In our domain, this data must be communicated to a lander, which in turn sends the results to a control center situated on Earth (current Mars rovers communicate to Earth via orbital relays). The communication from Mars to Earth has a long delay of at least 2.5 minutes, and at most 22 minutes. The rover could perform reconnaissance activities [12] to identify interesting targets and maps for future missions. After a long time, the rover may lose capabilities [28,42] (e.g. reduced mobility due to wear and tear on motors or mechanical systems, reduced power generation due to battery degradation, accumulated dust on solar arrays), as happened with the Spirit and Opportunity rovers.

Our interest is providing rovers with on-board execution and reactive planning capabilities so that they can repair plan failures by themselves in a timely fashion and thus reduce the communication overhead with the Earth. This capability is present in a limited way on current rovers for navigation and hazard avoidance [15], and experiments have been conducted involving replanning to perform opportunistic science. These capabilities will be needed for future missions. This future hypothetical Mars mission scenario is the problem that has primarily motivated our work.

7. Test and evaluations

Our domain-independent reactive model has been tested in a problem scenario that simulates the Mars domain described in Section 6 as well as in other domains such as supply-chain, manufacturing or vehicle routing. All of the domains and problems are encoded in PDDL. For the Mars domain, we adapted the PDDL files of the rovers domain from the International Planning Competition[6] (IPC) to endow rovers with reconnaissance abilities like, for instance, the operators (SeekSoil ?r ?w) or (SeekRock ?r ?w) that allows a rover ?r to seek more samples in a waypoint ?w. The resulting files accurately reflect the size and difficulty of the real problems except for the numeric capabilities of the rovers.

In this section we show the results obtained for 12 problems of the Mars scenario and 10 problems of a vehicle routing domain [45] adapted from the logistics domain of the IPC, a significant problem in the industry of transportation. For both domains, we discuss the accuracy of our approach to generate the repairing structures within the given time; additionally, we compare the average time to solve a plan failure in the Mars domain with our approach and other replanning and plan-adaptation mechanisms.

The reactive execution model was implemented in Java and all tests were run on a GNU/Linux Debian computer with an Intel 7 Core i7-3770 CPU @ 3.40 GHz × 8, and 8 GB RAM.

7.1. Time-bounded repairing structures

In order to test the timely generation of a repairing structure, we executed several problems of diverse complexity of the Mars exploration and vehicle domains, and we gathered the results of the first three generated $\mathcal{T}_i$ for each problem (see Table 2). The data shown for every $\mathcal{T}_i$ are: Π_A, number of remaining actions of the plan when $\mathcal{T}_i$ was created; t_s, the time limit to create $\mathcal{T}_i$; the values of l and d selected by the estimation model within t_s; $t_{\mathcal{T}}$, the real time used in the construction of $\mathcal{T}_i$ and N, the number of partial states in $\mathcal{T}_i$. All times are measured in seconds. Notice that the value of t_s for $\mathcal{T}_i$ is subject to the value of l in $\mathcal{T}_{i-1}$, except for $\mathcal{T}_1$, which is a fixed value of 1 sec. The number of locations, resources and goals contribute to increase the complexity of each problem.

The top part of Table 2 shows the results for the Mars exploration domain. Three repairing structures ($\mathcal{T}_1$, $\mathcal{T}_2$ and $\mathcal{T}_3$) were collected for all the problems except for problem 2, where $\mathcal{T}_1$ and $\mathcal{T}_2$ covered all the actions of Π_A. The first remark about these results is that almost every $\mathcal{T}_i$ was generated within the deadline ($t_{\mathcal{T}_i} < t_s$), excluding $\mathcal{T}_3$ of the problem 12 that slightly exceeded its time limit. A second observation is that, for relatively small problems, like 1–5, the value of $t_{\mathcal{T}}$ is far from t_s because the search space of these problems is fairly small and the newly generated partial states are all repeated nodes after a certain point. Hence, in most of the simplest problems the entire state space is quickly exhausted, in contrast to the problems 6–12. In some problems, we can also observe that the values of l, d and t_s are the same but the values of $t_{\mathcal{T}}$ and N are different. For instance, $(l, d) = (3, 4)$ and $t_s = 2$ secs. for problems 7, 8 and 9 but the values of $t_{\mathcal{T}}$ and N are fairly different because of the increasing complexity of these problems in the number of locations and goals, which implies a higher branching factor in each pro-

[6] http://ipc.icaps-conference.org/.

Table 2
First three repairing structures of Mars and vehicle routing domains

		Data set complexity						$\mathcal{T}_1$ ($t_s = 1$ s)				$\mathcal{T}_2$					$\mathcal{T}_3$				
	Problem	Rovers	Locations	Goals				Π_A	l, d	$t_{\mathcal{T}}$	N	Π_A	l, d	t_s	$t_{\mathcal{T}}$	N	Π_A	l, d	t_s	$t_{\mathcal{T}}$	N
				(soil)	(rock)	(image)		(actions)	(size)	(s)	(nodes)	(actions)	(size)	(s)	(s)	(nodes)	(actions)	(size)	(s)	(s)	(nodes)
Mars exploration domain	1	1	3	1	0	1		6	2, 4	0.061	140	4	2, 5	2	0.046	219	2	2, 6	2	0.058	492
	2	1	3	1	0	1		6	4, 5	0.026	88	2	2, 6	4	0.036	128					
	3	1	3	1	1	1		10	4, 5	0.073	625	6	5, 6	4	0.091	777	1	1, 3	5	0.015	71
	4	1	4	1	1	1		9	2, 4	0.039	358	7	2, 6	2	0.098	1726	5	3, 5	2	0.106	1470
	5	1	4	2	1	1		13	2, 5	0.086	962	11	2, 6	2	0.067	1354	9	3, 5	2	0.110	1636
	6	2	4	2	1	1		11	3, 5	0.821	2905	8	3, 5	3	0.565	3822	5	2, 6	3	0.217	1429
	7	2	4	3	1	1		16	3, 5	0.259	2065	13	2, 6	3	0.221	798	11	3, 4	2	0.297	1458
	8	2	5	2	2	1		16	2, 6	0.243	3965	14	2, 5	2	0.273	1911	12	3, 4	2	0.431	5578
	9	2	6	3	3	1		22	2, 6	0.942	13981	20	2, 5	2	0.461	4686	18	3, 4	2	0.990	8461
	10	2	6	3	3	2		25	2, 3	0.240	801	23	2, 4	2	1.103	9479	21	2, 4	2	0.984	9959
	11	4	10	2	4	3		35	2, 5	0.322	761	33	3, 4	2	1.509	6238	30	2, 4	3	1.105	4926
	12	4	10	3	4	3		39	2, 5	0.250	756	37	2, 4	2	0.272	1439	35	2, 4	2	2.125	12366
	Problem	Airport	Locations	City	Plane	Truck	Goals (packages)														
Vehicle routing domain	1	2	2	2	1	2	4	20	3, 6	0.285	341	17	4, 6	3	0.495	621	13	5, 6	4	0.034	120
	2	2	2	2	1	2	5	27	3, 6	0.140	314	24	4, 6	3	0.149	418	20	3, 6	4	0.137	433
	3	2	2	2	1	2	6	25	2, 6	0.131	262	23	2, 6	2	0.112	284	21	4, 6	2	0.382	683
	4	3	3	3	1	3	7	36	2, 6	0.448	421	34	3, 6	2	1.049	896	31	5, 6	3	0.160	223
	5	3	3	3	1	3	8	31	3, 5	0.457	861	28	5, 6	3	2.647	2070	23	5, 6	5	4.918	6573
	6	3	3	3	1	3	9	36	3, 5	0.527	915	33	2, 6	3	1.963	1427	31	2, 6	2	0.803	641
	7	4	4	4	1	4	10	46	4, 5	0.954	2054	42	4, 6	4	3.624	1612	38	4, 6	4	0.623	382
	8	4	4	4	1	4	11	50	3, 5	1.091	1344	47	3, 6	3	3.803	2355	44	5, 6	3	0.685	340
	9	4	4	4	1	4	12	42	2, 5	0.937	1164	40	4, 5	2	1.947	2965	36	2, 6	4	1.645	633
	10	5	5	5	2	5	13	81	3, 5	0.595	511	78	4, 5	3	1.977	1341	74	3, 6	4	4.973	1507

Table 3
Time and recovery plans for repairing tasks with $\mathcal{T}_1$ in the Mars exploration domain

	Problem	1		2				3				4		5		6			7			8		9		10		11		12	
	Failure id	1	2	1	2	3	4	1	2	3	4	1	2	1	2	1	2	3	1	2	3	1	2	1	2	1	2	1	2	1	2
	Type	D	A	A	B	A	A	D	A	A	D	A	B	A	A	A	A	C	A	B	A	A	A	A	B	A	A	A	A	A	C
	Π_A	6	5	6	5	4	3	10	9	8	7	9	8	13	12	11	10	9	16	15	14	16	15	22	21	25	24	35	34	39	38
Repairing (RP)	Reused Π_A	5	4	6	5	3	✗	8	9	8	6	9	8	11	12	11	10	8	14	15	14	16	15	21	21	25	23	33	34	39	38
	Π'_A	5	6	7	6	4	✗	9	10	9	6	10	9	15	13	12	11	11	18	16	16	17	16	24	22	26	25	37	35	41	39
			(root)			(root)		(root)			(root)			(root)				(root)			(root)			(root)			(root)	(root)			
	Time (ms)	1.03	0.60	0.16	5.62	1.02	0.81	0.28	0.11	0.10	1.45	0.29	0.26	5.22	0.18	0.26	0.22	0.95	1.39	0.12	2.12	0.27	0.24	23.35	0.21	0.48	2.94	4.28	0.43	0.71	0.34
Adaptation (LPG-ADAPT)	Reused Π_A	5	4	6	5	3	✗	10	8	8	4	9	7	10	10	11	10	9	13	14	12	15	14	18	19	23	21	30	27	29	33
	Π'_A	5	9	9	6	4	✗	13	11	9	6	12	12	13	13	13	15	14	18	19	14	19	20	23	25	28	25	40	37	45	41
	Time (ms)	54	45	58	52	43	51	45	44	43	53	53	52	55	45	48	45	49	56	49	57	43	53	44	56	44	45	51	49	52	50
Replanning (LAMA)	Reused Π_A	5	4	5	5	3	✗	6	4	4	5	5	6	9	7	3	1	2	4	10	0	4	4	13	4	15	3	22	25	21	29
	Π'_A	5	6	7	6	4	✗	11	9	9	5	11	11	15	15	12	10	14	17	17	14	18	16	24	24	28	27	36	34	39	42
	Time (ms)	36	39	54	38	36	32	39	38	41	39	41	41	42	59	51	51	51	51	52	52	59	57	68	69	74	72	140	166	148	149

blem. In general, the branching factor also depends on the *relevant variables* of each particular plan window and, hence, a tree like $\mathcal{T}_2$ of problem 12 may result in a smaller search space than $\mathcal{T}_2$ of problem 10.

The lower part of Table 2 shows the results obtained for the vehicle domain, where the goal is to find optimal routes for several vehicles which have to deliver a number of packages. Most of the repairing structures were generated within the time limit except in the case of $\mathcal{T}_1$ and $\mathcal{T}_2$ of the problem 8, and $\mathcal{T}_3$ of the problem 10, due to the complexity of these problems. In general, the trees in this domain are larger than those of the Mars domain under the same value of t_s. This is because the branching factor in the vehicle domain is much lower than in the Mars domain since trucks are confined to move in a particular city and, hence, loading a package in a truck will only ramify across the trucks defined in the city where the package is located. In contrast, in the Mars domain rovers are equipped with all functionalities and therefore the branching factor is considerably higher.

All in all, the results corroborate the accuracy of the predictive model and the timely generation of the repairing structures. Overall, out of all of the experiments carried out in the diverse domains, we can say that 95% of the repairing structures are always generated within the time limit. On the other hand, in contrast to most of reactive planners [1,13], our model is far more flexible since it is capable of dealing with more than one action (in the plan window) and multiple failure states with a single repairing structure.

7.2. Repairing plan failures

This section shows the performance of the plan recovery search procedure when repairing a plan failure in the Mars domain. Specifically, failures were randomly simulated in the plan window of the tree $\mathcal{T}_1$ for the problems in the upper part of Table 2. Failures were generated by changing the value of a fluent in the initial state and in the states resulting from the execution of an action a_i, thus provoking an erroneous execution in the next action a_{i+1} of the plan window. In other words, the first failure affects action a_1 of Π_A, the second failure affects a_2 of Π_A assuming a_1 was correctly executed, the third failure affects a_3 assuming the two preceding actions were correctly executed and so on.

Table 3 depicts the results of the repairing task in the Mars domain with three approaches: our RP, the LPG-ADAPT mechanism [18], which repairs a given plan adapting it to a new current state, and the classical planner LAMA [38], used as a deliberative planner to obtain a new plan from scratch (replanning) when a failure is detected. Plan failures are classified as follows:

A. Failures originated by an error in the execution of the actions (Navigate ?r ?w_o ?w_d). An error of this type is because of: 1) the rover ?r is not located in ?w_o at the time of executing the action or 2) the path from ?w_o to ?w_d is blocked and the rover cannot traverse it.
B. Failures that prevent the rover from analyzing the results or taking good pictures. This type of failure is caused by: 1) the rover loses the sample (rock or soil) by an unexpected event when it is about to analyze it, or 2) the camera loses calibration before taking the picture.
C. Failures that are solved with the help of other rovers when a hardware failure disables the device of a rover. Since fixing the damaged hardware is not an eligible option in the NASA scenario, the only possible way to repair this failure is with the help of other rover, as long as this is possible within the repairing structure.
D. Failures that positively affect the plan execution.

Table 3 also shows the number of remaining actions of the plan Π_A at the time of executing the repairing task, the number of actions reused from Π_A, the total number of actions of the recovery plan (Π'_A) and the time of the plan repair measured in milliseconds. The parameter *reused* Π_A represents the stability with respect to Π_A; i.e. the number of actions of Π_A that are reused in Π'_A. Finally, the failures labeled as root in the Π'_A row denote that the recovery plans were calculated up to the root node of $\mathcal{T}_1$.

As we can see in Table 3, the three approaches were able to find a plan except for failure 4 of problem 2. In this case, the path from waypoint w_1 to w_2 is blocked and there is no other possible way to navigate to w_2; that is, it does not exist a recovery plan for this failure.

Regarding stability, the solution plans found by the RP can be classified as: (1) plans that benefit from positive failures of type D, and then contain fewer actions than the original plan Π_A; (2) plans that reuse all the actions in Π_A (e.g. failure 1 of problems 2 and 6); and (3) plans that reuse only some of the actions in Π_A (e.g., failure 1 of problems 5 and 11).

The solutions to failures of type A are about rerouting rovers through other paths using the available travel maps. The recovery from failures of type B implies either exploring the area again seeking a new sample of rock or soil (e.g., failure 2 of problem 2) and then an-

Table 4
Summary of statistics for RP, LPG-ADAPT and LAMA performance

	Stability (%)		$\Delta\ \Pi_A$		Time (ms)	
	μ	σ	μ	σ	μ	σ
RP	92	19	0.97	0.85	1.85	4.33
LPG-ADAPT	85	19	2.40	1.94	49.47	4.72
LAMA	51	27	1.33	1.52	62.83	36.99

alyze it, or calibrating the rover's camera again (e.g., failure 2 of problem 4). Failures of type C were found in two cases, which could be repaired because their respective $\mathcal{T}_1$ included paths involving the second rover (e.g., failure 3 of problem 6). Here, a hardware failure prevent the rover from analyzing the soil in a specific location, our model repairs the failure using the second rover that explores the area seeking for soils, analyzes the soil and communicates the results to the lander. In the failures of type D, the RP takes advantage of the *positive* failure, which achieves the effects of the next action to execute and, consequently, the RP proceeds with the following action in Π_A.

The performance results in Table 3 and the summary of statistics in Table 4 show that our RP performs admirably well in all the measured dimensions. Regarding stability, RP outperfomrs LPG-ADAPT and LAMA. LAMA is the approach with the worst rate of stability (51%), which is reasonable since the planner does not repair a plan but it computes a new plan. In Table 3 we can see that the plan quality or number of actions of Π'_A is slightly higher with the RP than with LAMA in some cases (e.g., failure 1 of problem 12 or failure 3 of problem 7), and lower in some other cases (e.g. failure 2 of problem 12 or failures 1 and 2 of problem 10). LAMA is able to find shorter plans in a few cases because it computes a plan for the new situation without being subject to keep the actions in Π_A. Nevertheless, all in all, the RP returns plans of better quality than LAMA as Table 4 shows (the mean value in the increase of the number of actions is 0.97 in RP against 1.33 in LAMA). The comparison between RP and LPG-ADAPT clearly benefits RP in both stability and quality of the recovery plan, particularly in the most complex problems (10 to 12). As for the computation time, finding a recovery path to the root node is more costly since the repairing mechanism explores the entire search space. However, RP shows outstanding results compared to LPG-ADAPT and LAMA, which proves the benefit of using the RP to repair plan failures in reactive environments besides avoiding the overhead of communicating with a deliberative planner. In conclusion, we can affirm that our model is a robust recovery mechanism for reactive planning that also provides good-quality solutions.

8. Limitations and extensions of the model

The results in Section 7 show that our reactive execution model meets the performance needs of a reactive plan repair and that it also outperforms other repairing mechanisms. However, the model presents some limitations that we intend to overcome in the future. One limitation is the machine dependency of the estimation model explained in Section 5.3. In order to reproduce the experiments, or to export them to other systems, the training of the estimation model must be repeated to adjust the value of $\bar{t}_\Gamma$ for a particular processor.

Assuming several agents executing their plans in a common environment, a repairing task of an agent might cause conflicts in the plan of the others. Additionally, the occurrence of failures could restrict the capabilities of the agents, preventing them from achieving some goal. Therefore, a multi-agent approach where execution agents act, coordinate and jointly repair a failure is desirable [2,16,35]. A communication protocol that helps agents request, provide and agree on a particular recovery plan is a desirable approach for a multi-agent repair system [25]. Particularly, the present work represents a first step towards a multi-agent P&E system capable of coordinating agents plans while minimizing crowd-effects [44].

Our model can be easily extended to parallel and temporal planning. The parallel execution of several actions of an agent is achievable by grouping together the preconditions and effects of the parallel actions into a new action. The existence of grouped actions would avoid the duplicated states that arise from the multiple serialization of the actions in the group. On the other hand, handling durative actions in temporal planning would involve creating a regressed partial state at each relevant time point of an action and adding fluents to represent the ongoing executing actions at each execution cycle.

9. Conclusions and future works

This paper presents a reactive execution model, which comprises a RP, to recover from failures in planning control applications. The model is embedded into a P&E system where an execution agent receives a plan from a deliberative planner and its mission is to monitor, execute and repair the given plan.

Providing time-bounded responses in reactive environments is a difficult and sometimes unfeasible task

due to the unpredictability of the environment and the impossibility to guarantee a response within a given time. An alternative solution to overcome this difficulty is working with time-bounded data structures rather than designing time-bounded reasoning processes. By following this approach, our model ensures the availability of a repairing structure, or search tree, within a given time, which is later used to fix the action failures during the plan execution.

Several features have been considered in order to have a search tree generated in due time: (1) the tree is formed of partial states which contain far less fluents than world states; (2) the tree is limited to a particular fragment of the plan and tree depth that are calculated by an estimation model; and (3) the expansion of the tree only considers the relevant variables that might potentially fail during the plan execution. Under these criteria, we show the results obtained for two different domains, a simulation of a real NASA space problem, and a vehicle routing domain. The results corroborate that there is a 95% likelihood to obtain a repairing structure in time. Additionally, the exhaustive experimentation on the repairing tasks confirm that the repairing structure together with the search recovery process is a very suitable mechanism to fix failures that represent slight deviations from the main course of action in a planning control application. The results support several conclusions: the accuracy of the model to generate repairing structures in time, the usefulness of a single repairing structure to repair more than one action in a plan fragment while reusing the original plan as much as possible, and the reliability and performance of our recovery search procedure in comparison with other well-known classical planning mechanisms.

The current RP can be extended in several different directions as, for instance, by including the necessary machinery to deal with temporal plans. Our next future work is to exploit this model for a multi-agent recovery mechanism in which agents dynamically form a teamwork at execution time and work together in the repair of a plan failure.

Acknowledgments

This work has been partly supported by the Spanish MICINN under the projects TIN2014-55637-C2-2-R, and the Valencian project PROMETEOII/2013/019.

References

[1] P. Aschwanden, V. Baskaran, S. Bernardini, C. Fry, M. Moreno, N. Muscettola, C. Plaunt, D. Rijsman and P. Tompkins, Model-unified planning and execution for distributed autonomous system control, in: *AAAI Fall Symposium on Spacecraft Autonomy*, (2006).

[2] R. Badawy, A. Yassine, A. Heßler, B. Hirsch and S. Albayrak, A novel multi-agent system utilizing quantum-inspired evolution for demand side management in the future smart grid, *Integrated Comp-Aided Engineering* **20**(2) (2013), 127–141.

[3] A.G. Banerjee and S.K. Gupta, Research in automated planning and control for micromanipulation, *IEEE Transactions on Automation Science and Engineering* **10**(3) (2013), 485–495.

[4] P. Baraldi, R. Canesi, E. Zio, R. Seraoui and R. Chevalier, Genetic algorithm-based wrapper approach for grouping condition monitoring signals of nuclear power plant components, *Integrated Comp-Aided Engineering* **18**(3) (2011), 221–234.

[5] J.C. Bongard and H. Lipson, Automated damage diagnosis and recovery for remote robotics, in: *IEEE Robotics and Automation* **4** (2004), 3545–3550.

[6] M. Brenner and B. Nebel, Continual planning and acting in dynamic multiagent environments, *Autonomous Agents and Multi-Agent Systems* **19**(3) (2009), 297–331.

[7] B. Browning, J. Bruce, M. Bowling and M. Veloso, STP: Skills, tactics and plays for multi-robot control in adversarial environments, *IEEE Journal of Control and Systems Engineering* **219** (2005), 33–52.

[8] J. Bryson and L.A. Stein, Modularity and design in reactive intelligence, *Intl Joint Conference on Artificial Intelligence* (2001), 1115–1120.

[9] S. Chien, B. Cichy, A. Davies, D. Tran, G. Rabideau, R. Castaño, R. Sherwood, D. Mandl, S. Frye, S. Shulman, J. Jones and S. Grosvenor, An autonomous earth-observing sensorweb, *IEEE Intelligent Systems* **20**(2005), 16–24.

[10] J.Y.J. Chow, Activity-based travel scenario analysis with routing problem reoptimization, *Comp-Aided Civil and Infrastruct Engineering* **29**(2) (2014), 91–106.

[11] R.E. Fikes, P.E. Hart and N.J. Nilsson, Learning and executing generalized robot plans, *Artificial Intelligence* **3** (1972), 251–288.

[12] W. Fink, J.M. Dohm, M.A. Tarbell, T.M. Hare and V.R. Baker, Next-generation robotic planetary reconnaissance missions: A paradigm shift, *Planetary and Space Science* **53**(14) (2005), 1419–1426.

[13] A. Finzi, F. Ingrand and N. Muscettola, Model-based executive control through reactive planning for autonomous rovers, in: *IEEE Intelligent Robots Systems* **1** (2004), 879–884.

[14] L. Flückiger and H. Utz, Service oriented robotic architecture for space robotics: Design, testing, and lessons learned, *J of Field Robotics* **31**(1) (2014), 176–191.

[15] T.W. Fong, M. Bualat, M. Deans, M. Allan, X. Bouyssounouse, M. Broxton, L. Edwards, R. Elphic, L. Fluckiger, J. Frank, L. Keely, L. Kobayashi, P. Lee, S.Y. Lee, D. Lees, E. Pacis, E. Park, L. Pedersen, D. Schreckenghost, T. Smith, V. To and H. Utz, Field testing of utility robots for lunar surface operations, in: *AIAA Space Conference and Exposition* (2008), 22–27.

[16] A. Fougères and E. Ostrosi, Fuzzy agent-based approach for consensual design synthesis in product configuration, *Integrated Comp-Aided Engineering* **20**(3) (2013), 259–274.

[17] M. Fox and D. Long, Pddl2.1: An extension to pddl for expressing temporal planning domains, *Journal of Artificial Intelligence Research*, **20** (2003), 61–124.

[18] M. Fox, A. Gerevini, D. Long and I. Serina, Plan stability: Replanning versus plan repair, in: *Automated Planning and*

Scheduling (2006), 212–221.

[19] C. Fritz and S.A. McIlraith, Monitoring plan optimality during execution, in: *Automated Planning and Scheduling* (2007), 144–151.

[20] A. Garrido and E. Onaindia, Assembling learning objects for personalized learning: An ai planning perspective, *IEEE Intelligent Systems* **28**(2) (2013), 64–73.

[21] M. Ghallab, D. Nau and P. Traverso, *Automated Planning: Theory & Practice*, Elsevier, 2004.

[22] M. Ghallab, D.S. Nau and P. Traverso, The actor's view of automated planning and acting: A position paper, *Artificial Intelligence Journal* **208** (2014), 1–17.

[23] K. Görlach, M. Sonntag, D. Karastoyanova, F. Leymann and M. Reiter, Conventional workflow technology for scientific simulation, in: *Guide to e-Science* (2011), 323–352.

[24] C. Guzmán, V. Alcázar, D. Prior, E. Onaindia, D. Borrajo, J. Fdez-Olivares and E. Quintero, PELEA: A domain-independent architecture for planning, execution and learning, in: *Scheduling and Planning Applications woRKshop* **12** (2012), 38–45.

[25] C. Guzmán, P. Castejon, E. Onaindia and J. Frank, Multi-agent reactive planning for solving plan failures, in: *Hybrid Artificial Intelligent Systems – 8th International Conference*, Lecture Notes in Computer Science **8073** (2013), 530–539.

[26] M.R. Hagerty and V. Srinivasan, Comparing the predictive powers of alternative multiple regression models, *Psychometrika* **56**(1) (1991), 77–85.

[27] R. Haijema and E.M.T. Hendrix, Traffic responsive control of intersections with predicted arrival times: A markovian approach, *Comp-Aided Civil and Infrastruct Engineering* **29**(2) (2014), 123–139.

[28] Y. Kuwata, A. Elfes, M. Maimone, A. Howard, M. Pivtoraiko, T.M. Howard and A. Stoica, Path planning challenges for planetary robots, in: *IEEE Intelligent Robots Systems* (2008), 22–27.

[29] S. Lemai and F. Ingrand, Interleaving temporal planning and execution in robotics domains, in: *Innovative Applications of Artificial Intelligence* (2004), 617–622.

[30] M.W. Maimone, P.C. Leger and J.J. Biesiadecki, Overview of the mars exploration rovers' autonomous mobility and vision capabilities, in: *IEEE Robotics and Automation*, Jet Propulsion Laboratory, NASA (2007).

[31] M.G. Marchetta and R. Forradellas, An artificial intelligence planning approach to manufacturing feature recognition, *Comp-Aided Design* **42**(3) (2010), 248–256.

[32] C. McGann, F. Py, K. Rajan, H. Thomas, R. Henthorn and R. McEwen, A deliberative architecture for auv control, in: *IEEE Robotics and Automation* (2008), 1049–1054.

[33] N. Meuleau and D.E. Smith, Optimal limited contingency planning, *Uncertainty in Artificial Intelligence* (2003), 417–426.

[34] A. Milani and V. Poggioni, Planning in reactive environments, *Computational Intelligence* **23**(4) (2007), 439–463.

[35] I. Montalvo, J. Izquierdo, R. Pérez-García and M. Herrera, Water distribution system computer-aided design by agent swarm optimization, *Comp-Aided Civil and Infrastruct Engineering* **29**(6) (2014), 433–448.

[36] J. Patel, M.C. Dorneich, D.H. Mott, A. Bahrami and C. Giammanco, Improving coalition planning by making plans alive, *IEEE Intelligent Systems* **28**(1) (2013), 17–25.

[37] C. Piacentini, V. Alimisis, M. Fox and D. Long, Combining a temporal planner with an external solver for the power balancing problem in an electricity network, in: *the 23th Automated Planning and Scheduling*, AAAI (2013), 398–406.

[38] S. Richter and M. Westphal, The LAMA planner: Guiding cost-based anytime planning with landmarks, *Journal of Artificial Intelligence Research* **39**(1) (2010), 127–177.

[39] A. Rodríguez and J.A. Reggia, Collective-movement teams for cooperative problem solving, *Integrated Comp-Aided Engineering* **12**(3) (Jul 2005), 217–235.

[40] M. Santofimia, X. del Toro, P. Roncero-Sánchez, F. Moya, M. Martinez and J. López, A qualitative agent-based approach to power quality monitoring and diagnosis, *Integrated Comp-Aided Engineering* **17**(4) (2010), 305–319.

[41] J. Sedano, C. Chira, J. Villar and E. Ambel, An intelligent route management system for electric vehicle charging, *Integrated Comp-Aided Engineering* **20**(4) (2013), 321–333.

[42] M. Smart, B. Ratnakumar, L. Whitcanack, F. Puglia, S. Santee and R. Gitzendanner, Life verification of large capacity yardney li-ion cells and batteries in support of nasa missions, *Intl Journal of Energy Research* **34**(2) (2010), 116–132.

[43] S. Sohrabi, J.A. Baier and S.A. McIlraith, Diagnosis as planning revisited, in: *21st Intl Workshop on the Principles of Diagnosis* (2010),

[44] Q. Sun and S. Wu, A configurable agent-based crowd model with generic behavior effect representation mechanism, *Comp-Aided Civil and Infrastruct Engineering* **29**(7) (2014), 531–545.

[45] P. Toth and D. Vigo, *The Vehicle Routing Problem*, Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 2001.

[46] C. uise, J.C. Beck and S.A. McIlraith, Flexible execution of partial order plans with temporal constraints, in: *the 23th Intl Joint Conference on Artificial Intelligence*, AAAI, (2013), 2328–2335.

[47] W. Xie and Y. Ouyang, Dynamic planning of facility locations with benefits from multitype facility colocation, *Comp-Aided Civil and Infrastruct Engineering* **28**(9) (2013), 666–678.