Journal of Artificial Intelligence Research X (2016) X-XX Submitted 8/16; published X/XX

Motivation-Based Hierarchical Probabilistic Planning with Resources for an Autonomous Robot Deliberative System

Raphael Gottstein GOTTSTEIN@ISIR.UPMC.FR
Mihai Andries ANDRIES@ISIR.UPMC.FR
Raja Chatila RAJA.CHATILA@ISIR.UPMC.FR
Sorbonne Universités, UPMC Univ Paris 06, UMR 7222, Institut des Systèmes Intelligents et de Robotique, F-75005, Paris, France

Abstract

In this work, we aim to build a decisional architecture that can handle multiple and concurrent goals, as well as resources, under uncertainty and choose autonomously how to satisfy these goals. We introduce an original approach to represent goals, that we call motivations, represented as finite state machines. This representation allows a much faster (but sub-optimal) solution search, introducing structure into the solution search space. Motivation states evolve through policies provided by MDPs. Then, by computing precisely the effects of those policies on the world, we produce a new higher level model of the problem. In this model, the computed policies are akin to macro-actions. Finally, we use a local solver to find the best way to link policies and maximise rewards defined by the motivations for a given time horizon.

1. INTRODUCTION

We want to build a system able to reach simultaneous, potentially concurrent goals that have different priorities, and to manage resources such as energy and time, in a stochastic world. We also aim for autonomous initiative and decision-making, so that the agent does not only react to particular stimuli or direct requests from a human, but also chooses by itself goals to achieve. Knowing the effects of resolution of goals, both on the world states and on the resolution of other goals, is critical for choosing autonomously between goals.

In the domain of probabilistic decision-making, the decision process tries to maximise an *utility* outcome, by obtaining *rewards* or *penalties* attached to the actions that the agent can undertake and to the states in which the agent may find itself. These rewards/penalties are given by a *reward function*, which is constructed in such a way, so as to push the agent to reach a specific goal.

To optimally solve the problem of decision making, a reward function is used, which treats/considers all the objectives jointly. However, processing this joint objective makes the search space too large for a solution to be found in reasonable time (e.g. if we want an agent to do A, then B and finally C, we are forced to introduce state variables to represent the problem so as to express "*what is the agent's next objective*"). A sub-optimal alternative for solving the problem of decision making consists in decomposing the objectives into separate sub-problems, in order to be able to process each search space in a reasonable time. However, in this case the difficulty lies with predicting the behaviour of the different solutions and the possible interactions between the solutions to these sub-problems, and

chaining together these solutions (e.g. compute separately "do A", "do B", "do C", and manage to chain the three sub objectives). In this paper, we will adopt this last option.

The main idea behind our approach is to introduce the notion of *motivation*, as a structure consisting of (individual or chained) goals, which may be permanently active or not, and to which we associate rewards. We aim to predict the precise effects of the resolution of a goal on the world and on other motivations, in order to compute a high-level plan, employing goal-reaching *policies* in the same way that we usually use *actions* in an Markovian Decision Process (MDP).

We thus propose an architecture that:

1. handles motivations,
2. computes possible policies for each motivation,
3. predicts the behaviour of each policy and its effect on motivations,
4. predicts the effects of a chain of policies,
5. finds an optimal arrangement of these policies, maximising the sum of the rewards obtained by the related motivations for a given time-horizon.

This architecture includes:

- an operational mechanism (world model, planner, etc.),
- an intentional model (the motivations), and
- a deliberation mechanism.

The remainder of this paper is structured as follows. In the next section, we overview the state of the art. In Section 3 we discuss the notion of *motivation* that we use to represent goals. Section 4 presents the global architecture and the operation of its components. Section 5 provides a complete description of the system on an assembly task example and experimental results. We conclude with Section 6.

Note: all notations used in this paper are summarised in a glossary at the end of the paper.

2. STATE OF THE ART

The problem of probabilistic decision with multiple goals and additional management of multiple resources is not addressed as a whole in the literature. However, there is a profusion of partial answers, that we can classify into:

- deterministic deliberation systems, sometimes with resource management,
- resource-less probabilistic planners.

For each category, we present related work and the planning methods they implement.

2.1 DETERMINISTIC DELIBERATION SYSTEMS

Deliberation systems exist for solving problems with multiple goals. They were applied in various environments, such as: manufacturing, industrial mobile robotics, exploration and rescue, service and domestic robots, autonomous spacecrafts, autonomous aerial vehicles, and autonomous cars.

Some deliberation systems consider resources in their planning, such as time (Georgeff & Lansky, 1987) and (Ghallab & Laruelle, 1994; Lemai & Ingrand, 2004) for modelling time-dependent goals, or energy level (Rabideau, Knight, Chien, Fukunaga, & Govindjee, 1999). This allows them to express different types of goals.

These deliberation systems have limited goal-management capacities. Most of them try to solve all goals jointly, generating a search space which is too large to explore in reasonable time (if the number of goals is considerable). To find a solution, most deliberation systems use search heuristics. Then, they use a plan-and-repair strategy to overcome the fact that the execution of deterministic plans often fails in practice. Another problem appears when conflicting goals are given to the planner, in which case no suitable solution may be found. In (Pollack & Horty, 1999), such goal collisions are detected and eliminated by dropping out one of the conflicting goals.

Since an agent does not follow all of its goals at the same time (e.g. some goals become active only after completing other goals), it requires a method for structuring its goals. Yet, the literature remains unclear on how to represent chained goals' activation conditions. It is neither clear under which conditions a goal can be activated and sent to the planner, nor how it could impact the current plan (e.g. in (Muscettola, Nayak, Pell, & Williams, 1998) new goals are appended at the end of the current plan), nor how to manage goals' priorities in the deliberation and planning processes.

A solution to the problem of structuring goals is proposed in (Coddington & Luck, 2004), where goals are modelled as *drives*. They address problems with multiple resources, such as time, energy, or data storage (each drive is associated to a resource). A *drive* is a simple way to clearly define when a goal is active or not. Being associated with a resource (which changes value depending on the executed actions), a goal becomes active when the resource value crosses a pre-set threshold. By knowing the impact of each action on the amount of resources left, the system can predict when and which goals become activated, and thus plan future goals in advance (Coddington, 2007). Apart from the deterministic modelling of the world, this work appears to be close to our own. Nevertheless, while the way how a drive models an objective is interesting, it remains limited to goals that are turned on or off depending on a resource level. We consider that the activation condition of a goal should not be limited to resource thresholds. Moreover, in its current form, a drive cannot encode goal chaining. Regarding the planning part, the planner does not choose the order in which to resolve its goals, only aiming for the shortest plan. The consequences of a generated plan on the resources and drives activation are not taken into account in order to favour desired states. Consequently, a plan which will consume all battery left (or leaving a small amount causing to be unable to charge the battery again) could be generated.

In conclusion, deterministic deliberation systems provide several interesting tools to model and plan goals while managing resources. However, these systems employ deterministic methods of heuristic search and plan-repair, and thus work on an inaccurate model of

the environment. This renders it impossible to correctly predict the result of plan execution. How to properly structure, integrate, and manage goals still remains an open research question (Ingrand & Ghallab, 2014).

2.2 PROBABILISTIC APPROACHES

The world in which a robot evolves is too intricate to build an exact model of it. A convenient way to take into account uncertainty is to model the world using a stochastic representation. One solution is to represent the world roughly, not precisely enough to take every detail into account, and by transforming this lack of accuracy into uncertainty. Probabilistic methods are stochastic methods in whichan uncertainty outcome for an action is valued by the probability of obtaining this outcome. More realistic than deterministic methods, they can provide complete plans, called policies, and predict some policies' behaviour (Kolobov, 2012). However, in order to compute a complete solution for a problem, they need to enumerate every possible state, making complex problems impossible to solve in a reasonable amount of time.

We found very few references in the literature for the class of problems that we aim to address, i.e. probabilistic problems with multiple-goals and multiple-resource management Several planners can handle time (Mausam & Weld, 2008) and multiple resources (Bresina, Dearden, Meuleau, Ramakrishnan, Smith, et al., 2002; Guestrin, Hauskrecht, & Kveton, 2006), but most of them aim for a limited number of goal.

The work described in (Meuleau, Benazera, Brafman, Hansen, & Mausam, 2009) treats a class of problems very similar to ours. Their goal is to provide a planner for a Mars rover able to compute a conditional plan for resolving tasks when communication with Earth is not possible. The scenario includes a large set of goals, each promising a science return value (i.e. a reward value). With a limited amount of resources (time and energy), the planner must find a plan that maximises the science return value accumulated along the run. To explore the huge state space they face, they use the resources constraints to determine which state should be explored. However, the robot does not have the possibility to renew its resources and recharge its battery. So it has to select the best plan that could be accomplished respecting the time and energy constraints only once, which corresponds to optimise the run length.

Finally, the problem language RDDL (Relational Dynamic Influence Diagram Language) (Sanner, 2011) addresses this class of problems, currently without solvers able to process all the expressive capacity of the language (to our knowledge).

To reduce the computation time needed to find a solution, methods have been proposed that introduce structure into the problem:

- *options* (Sutton, Precup, & Singh, 1999), that compute short plans and use them as actions, in the same way as a Hierarchical Task Network (Nau, Au, Ilghami, Kuter, Murdock, Wu, & Yaman, 2003) does for deterministic problems,
- *task hierarchies* (Dietterich, 2000), which create macro-actions (similar to options) from the resolution of sub-goals,
- *factored Markovian Decision Processes* (Boutilier, Dean, & Hanks, 1999).

Options are a way to encode handmade routines (Sutton et al., 1999) that the agent will recurrently use in order to complete its mission. From an MDP point of view, an option is a handmade policy invokable in a limited set of states, for which is computed the probability of ending in each state and the expected reward, given a starting state. Thus, options can be considered as actions and used by search algorithms.

The *task hierarchy* method consists in resolving a set of objectives, composed of sub-goals, by computing sub-tasks (Dietterich, 2000) (i.e. plans for solving sub-goals). The idea is to determine which sub-task should be executed in which situations. Sub-tasks are computed separately to create task-resolving actions, akin to options. The main policy will then be computed using only sub-tasks as available actions. In (Dietterich, 2000), they propose to resolve each sub-task by defining terminal states and sub-reward functions (a local reward function specific to the sub-goal accomplishment).

Options and task hierarchies are interesting for the way they create high-level operators to subsequently compute a solution. These methods introduce structure into the problem, that reduces the number of available actions. This allows to reduce computational time and space, thus enabling to compute larger problems in reasonable time. Note that these methods do not guarantee the optimality of the solution, due to the decomposition of the problem. Hence, we talk about *hierarchical optimality* (Kolobov, 2012). However, none of these approximation techniques can currently handle resource management, as we propose to do.

We saw that a problem can be decomposed into sub-problems. Now, let us see how to use regularities within an MDP to optimise its resolution.

An *extensional representation* of the state space is a fully explicit enumeration of all possible states. On the other hand, an *intentional representation* only describes these states by using a set of features and their properties. In the *extensional representation*, when defining actions using a complete transition table, the actions which depend and act only on a subset of features will have redundant definitions, generated by the presence of the remaining independent features.

Factored MDP techniques (Boutilier et al., 1999) use an *intentional* representation of the world model, and propose to eliminate the aforementioned redundancies using factorised definitions. Factoring techniques allow to represent state spaces, transition functions, and reward functions in a factored way. Exploiting problem regularities allows to compute factored policies, and thus to reduce computation time. One solution to represent factored problems is to use *Probabilistic STRIPS Operators (PSOs)* (Boutilier et al., 1999). This is a standard representation inspired by STRIPS description language for deterministic problems.

By structuring problems, those techniques allow to reduce computation time. However, none of them implements resource management.

2.3 CONCLUSION ON THE STATE-OF-THE-ART

To conclude, we reviewed the state of the art in two related domains: deterministic deliberation systems and probabilistic planning, each capable of only partially solving the full problem of handling multiple goals with resource management under uncertainty.

The deterministic deliberation systems seen in the literature provide solutions for multi-goal planning, and some of them offer methods for modelling goals. However, the deterministic paradigm is not sufficient to model the world correctly, since it needs frequent plan repair due to unpredictable events and outcomes of actions. This limits the prediction capacity of these systems, and, in our opinion, renders it difficult for a robot to become autonomous.

On the other hand, while probabilistic description languages do exist to model our problem, no solver exists to propose complete solutions of it. Current solvers provide non-complete solutions (i.e. reach a goal state given a starting state), in contrast with complete solutions (i.e. that give a solution from every possible starting state). Since these techniques selectively develop the tree of reachable states, the resulting policy is incomplete, thus impeding the prediction of the solution's behaviour.

Nevertheless, several techniques exist to solve large problems, and offer complete solutions, and consequently a good prediction capability. Those techniques propose to decompose the problems and to structure them in order to take advantage of regularities within the problems. However, none of these techniques have been implemented to manage multiple resources at the same time.

In this paper, we propose a deliberation system for solving probabilistic, multi-goal and resource-dependent problems. In addition, we introduce a novel way to structure multiple goals into *motivations*, that we model as state-machines.

3. OBJECTIVES AS MOTIVATIONS

In the literature on probabilistic planners, an objective is usually described as a goal state, associated to a reward value, obtained when this goal state is reached. Some intermediate rewards can also be defined to guide the resulting policy to the given objective. Once the goal state is reached, the objective is consumed, and a new objective has to be given.

From our point of view, there are some objectives that cannot be rightfully expressed with this technique. For example, the objective "maintain a high battery level" is an objective that should not disappear when the battery level is high. We would want that objective to be deactivated when the battery level is high, and activated otherwise. It would also be convenient to know, when resolving a complex objective while having a high battery level, the probability of the battery level to become low, thus activating the objective of maintaining a high battery level.

Another particularity of objectives that classical goal states in probabilistic planning cannot correctly express is *chaining*. Take for example the objective "activate device A and then device B, in that order". This objective needs to differentiate its two steps and thus cannot be expressed by a single goal state. A workaround is to add a variable in the state representation of the problem, reflecting the progress of the goal, for example the variable "objectiveActivateDevices", which takes one of the values in $\{currentIsA, currentIsB\}$. The drawback of this technique is that it increases the size of the state space, and with it the time needed to converge to a solution. This makes it intractable to solve problems with multiple chained goals. Despite this workaround, it would be interesting to express chained goals in a proper manner, without additional state variables.

We propose to model objectives, including chained objectives, using *finite state machines*. We call this representation of objectives a *motivation*, to differentiate it from other representations. The next sections will describe the way these *motivations* work.

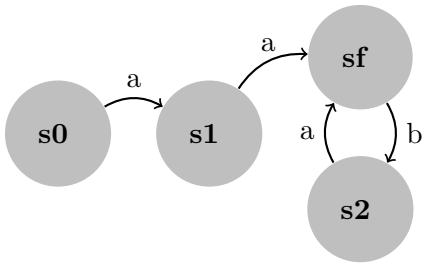


Figure 1: A finite state machine recognising the regular expression $aa(ba)^*$

3.1 MOTIVATION FINITE STATE MACHINE

In order to model permanent objectives, like "maintain a high battery level", or chained objectives, like "activate device A and then device B, in that order", we use finite state machines.

A finite state machine is composed of states and transitions. They are, for example, used to recognise regular expressions. Figure 1 shows an automata, that, starting from state $s0$ and finishing on state sf, recognises the regular expression $aa(ba)^*$.

In the same fashion, we want to change the state of a motivation (ms) when there is a **relevant** change in the state of the world (see the glossary for a summary of notations). World changes that do not impact the motivation state are ignored. For example, when the robot's battery level changes state, we also want the motivation state to change. In the example with device activation, the motivation state should change when the device A becomes activated. To check if the conditions are met for changing the motivation state, an observation of world transitions (ws, a, ws') is required, where world state (ws) is the initial world state, a is the executed action, and ws' is the resulting world state. World transitions provide information that can trigger changes in motivation states. This information allows to detect if: a given state was reached, a given action has been executed, a specific action was used in a given world state, etc. A component of (ws, a and ws') could possibly be defined as "any" or $*$, meaning that, in this case, any state or action will fit. These conditions that dictate changes in motivation states (from motivation state ms to ms') are called *motivation transitions*. For now, a *motivation transition* can be defined as an expression: $(ms, (ws, a, ws')) \rightarrow ms'$. In Section 3.1, we will add rewards to this representation.

Figure 2 shows two examples of motivations. For readability reasons, a short description of the world transitions is used in place of the true world transitions. The battery motivation in Figure 2a has two states, named $ms1$ and $ms2$, corresponding respectively to low and high battery levels. The motivation state changes from $ms1$ to $ms2$ (resp. $ms2$ to $ms1$) when the battery level changes from low to high (resp. from high to low). For the "device activation" motivation shown in Figure 2b, we have three states $ms1$, $ms2$ and $ms3$. They represent the states of the objective: "device A must be activated", "device B must be activated" and

"all activated". The motivation state changes from $ms1$ to $ms2$ when device A is activated, and from $ms2$ to $ms3$ when device B is activated.

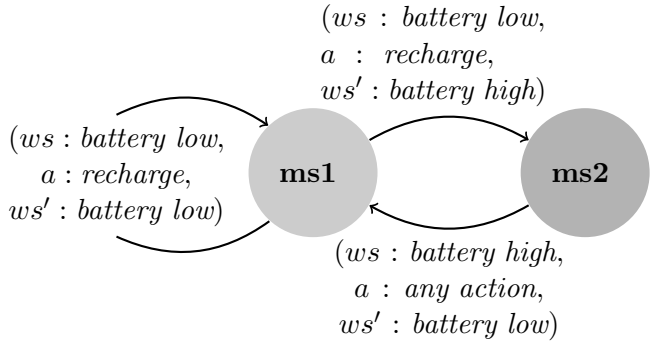
3.2 REWARDED MOTIVATION TRANSITIONS

In order to increase the expressiveness of motivations, we associate a reward to each motivation transition, reflecting its importance (Figure 3). A rewarded motivation transition (rmt) starting from $ms1$ and leading to $ms2$ is called an *available-rmt* when the current motivation state is $ms1$. It *becomes activated* (or *triggered*) when the corresponding world transition (ws, a, ws') happens, changing the motivation state to $ms2$ and obtaining the corresponding positive or negative reward (r). Figure 4 shows the motivations from the aforementioned examples, completed with rewards.

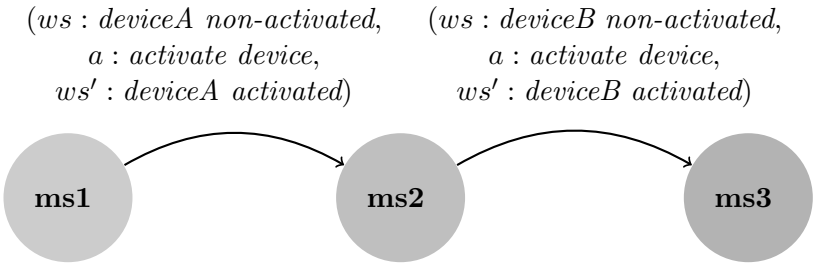
The notion of rmt will be intensively used in the rest of this paper. Rewarded motivation transitions will indeed allow to express the rewards of world transitions. The maximisation of the sum of these rewards will be sought by the deliberation system.

4. ARCHITECTURE OF THE DELIBERATIVE SYSTEM

Several architectures were proposed to organise robot decision making systems (or robot control architectures) to enable both reactive and deliberative capacities. One of the most widely used ones is the three-tiered or three-level architecture (Gat, 1992; Alami, Chatila,



(a) Battery management example



(b) Device activation example

Figure 2: Motivation examples for (a) battery objective, and (b) device activation objective.

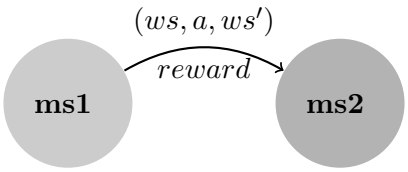
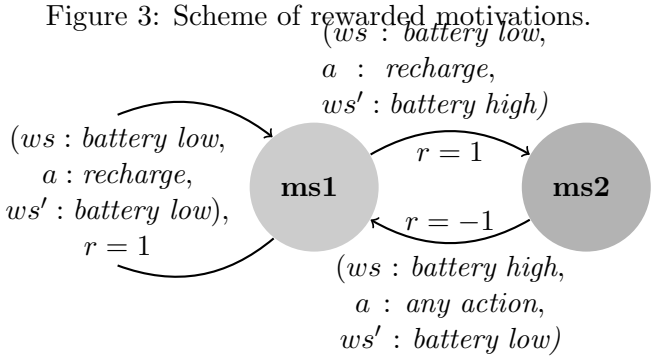
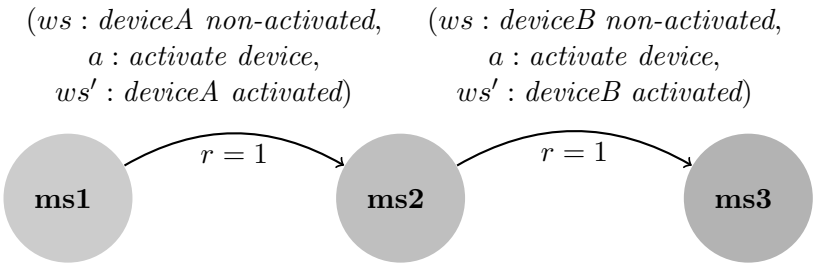


Figure 3: Scheme of rewarded motivations.



(a) battery management example



(b) device activation example

Figure 4: Examples of motivations with rewarded transitions.

Fleury, Ghallab, & Ingrand, 1998; Muscettola et al., 1998; Nesnas, Wright, Bajracharya, Simmons, & Estlin, 2003), which is composed of :

- A decision level composed of a planner and a time-bounded supervision system that allows (time-consuming) task planning, while enabling reactivity at the same time.
- An execution control level which controls and coordinates the execution of the planned tasks by the operational robot modules, satisfying logical and temporal constraints.
- A functional level which includes these modules, encapsulating robot functions and the middleware enabling them to communicate.

Figure 5 represents the higher level, such as in (Alami et al., 1998), which is of interest to us here, while the other levels are collapsed in the "robot" component. The supervisor embeds pre-defined procedures and a selection mechanism for solving simple situations. It takes the goals given by a user, and determines if its procedures can solve them, based

on its current goals and world state. If not, it asks the planner to compute an adequate plan. Therefore, the planner is only used when the supervisor does not have an action or procedure ready to achieve the goal. The supervisor handles action execution.

Our architecture replaces this higher level (supervisor-planner tandem). It is composed of three main components as illustrated in Figure 6:

- The *intentional module*, which manages the agent's objectives in the form of *motivations*.
- The *operational module*, which computes policies based on motivation automata, and computes predictions on resulting policies,
- The *deliberation module*, which is the main process. Its role is to provide to the operational module rewarded world transitions (*rwt*) to reach, to enable it to build predictable solutions that will trigger the corresponding rewarded motivation transitions (*rmt*). The deliberation module then computes the effects of these policies on the world state and on all motivations. These policies are used as macro-actions, to compute a conditional high-level plan for maximising the sum of the motivation rewards. This plan is called *policy agenda*.

In the literature, most of the decision systems that aim for "maximum average reward" are aiming in fact for *maximum average reward obtained per action/task executed*, which is the case for the classic MDP. Our actions' model includes the resources necessary for the achievement of actions. We treat time as a resource, and we can predict the amount of time required for each policy, therefore we are able to compute the average time needed to obtain a reward. Thus, we can build an architecture that aims to maximise the average reward obtained for a given time horizon (which is more realistic), rather than the reward obtained for a given horizon in terms of number of executed actions.

We next describe each component of the architecture.

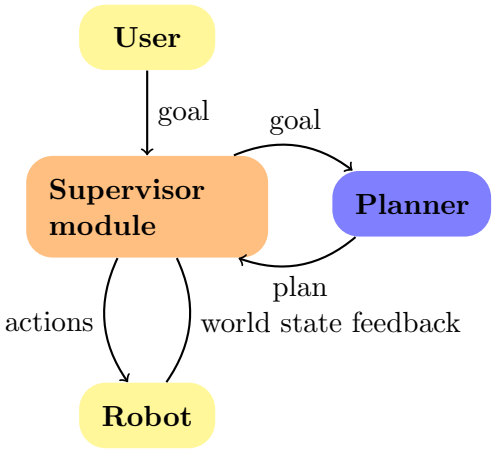


Figure 5: Architecture with a supervisor module for action planning and control.

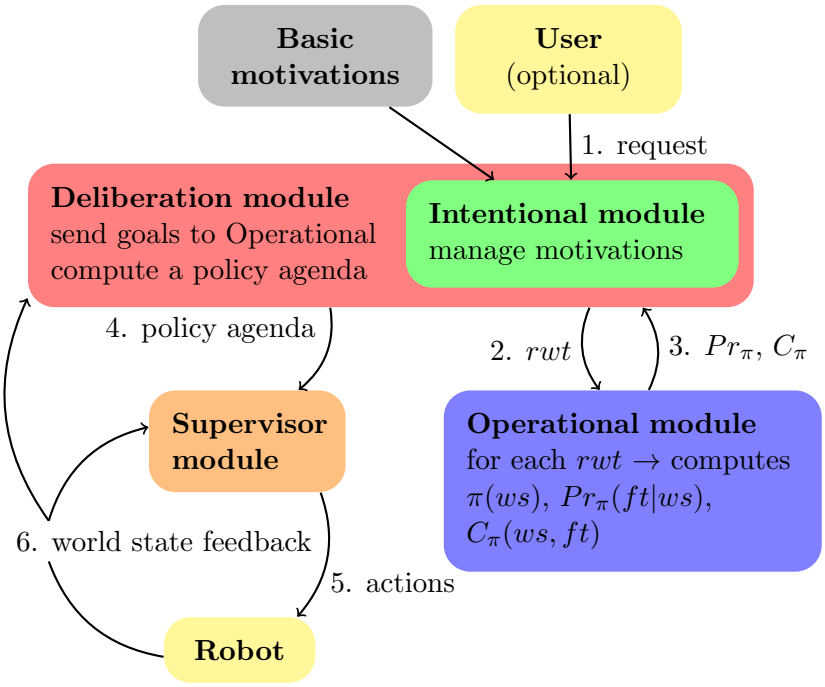


Figure 6: Proposed deliberative architecture. See related sections for a description of each component.

4.1 OPERATIONAL MODULE

This module is responsible for computing the policies to reach *rmt*s (given by the deliberation module) by using an MDP. It contains the model of the world: world and robot state space, actions space, transition function and resource costs defined using a Probabilistic STRIPS Operator-like representation. Figure 7 illustrates an example action modelled with this technique. The first line gives the action name. The second and the third lines give respectively the names of the variables on which the action depends, and the variables that can be changed by this action. Then, there can be multiple lines of preconditions. Each precondition is a conjunction of disjunctions, where each disjunction expresses a condition on the state of a given variable. If at least one precondition is met, the action can be used. Finally, the rules have a set of conditions and provide a set of consequences. If a condition is met, its corresponding effect is added to the possible outcomes of the action when being executed, with the given probability. The conditions apply on all the *condition variables* described in line 2. The effect applies on each *effect variable* described line 3, with the corresponding resource cost.

The PSO-like representation allows us to generate probabilistic transition tables on the fly, which correspond to the model of the world, and which are needed for an MDP to operate. We can also use a subset of such PSOs to build so-called *sub-models*. A sub-model is a reduced model of the world, with less variables (and thus a smaller world state space), and less actions. The idea is to verify, using the PSO-like representation of actions, that the

```
(action : putObject
condition variables : rP,oS,hS
effect variables : oS,hS
preconditions : (rP in {'A','B','C'}, oS in {'robotHand'}, hS in {'full'})
rules :
        (rP in {'A'}, oS in *, hS in *) -> ((oS='inA',hS='free'), energy= -2 time= +5, 0.85)
        (rP in {'A'}, oS in *, hS in *) -> ((oS='robotHand',hS='full'), energy= -3 time= +8, 0.15)
        (rP in {'B'}, oS in *, hS in *) -> ((oS='inB',hS='free'), energy= -2 time= +4, 1.00)
```

Figure 7: Example of a PSO-like partial representation of an action for putting an object. This can be read as "The action putObject takes rP (robot position), oS (object status) and hS (hand status) as condition variables and can change the state of the variables oS and hS. In order to execute this action, the rP value must be either A, B or C, the object must be in the robot hand, and hS must be full." This action model defines the world transition: (A, robotHand, full), putObject $\rightarrow$ (A, inA, free), with a probability of 0.85, a duration of 5 units of time and consumes 2% of energy. Similarly for robot position B, with a probability of 1.

selected actions do not depend on, or cannot alter other variables except for the selected ones. If this condition is verified, the sub-model can be generated and used in our MDP. A policy generated with a sub-model is defined on a reduced set of world states, which means that the generated policy does not take into account all variables. Nonetheless, since the available actions in the sub-model (and, therefore, in the policy) guarantee not to use or alter non-selected variables, the policy can be applied on a reduced state of the world.

This method allows to compute a policy which does not need all defined actions to be executed, but rather the subset of concerned variables and actions. This accelerates the computation in comparison to methods using the complete model. The set of PSOs selected for the computation of a policy is extracted from the *rmt* definitions. In the remainder of the paper, we will use sub-models for all operations concerning rewarded motivation transition policies, and for all computations regarding policies behaviour.

A rewarded world transition (rwt) is defined as a couple $(ft, reward)$. ft is a final transition: $ft = (ws, a, ws')$ represents the passage from the world state ws to ws' by executing action a and terminating the policy. The $reward$ is an MDP reward. A rwt is defined from an rmt, by keeping the final transition and $reward$ elements. The final transition represents the transition that the rwt aims to execute. These $(ft, reward)$ couples describe the reward function R, that we want to maximise in order to obtain the desired policy: $R(ft) = reward$ or $R(ws, a, ws') = reward$.

The operational module uses a customised MDP, which receives as input a rwt and computes a corresponding policy π_{rwt}, using the Value Iteration algorithm. Once calculated, the policies are analysed by a solver to predict their behaviours by computing respectively $Pr_\pi(ft|ws)$, the probability to finish by executing ft starting from the world state ws and using policy π_{rwt}, and $C_\pi(ws, ft)$, the resource costs for all resources when finishing on ft starting from ws applying π. For these two computations, the operational module uses the *sub-model* corresponding to the provided rwt instead of the complete model. The specific values of the Pr and C functions are predicted by solving equation systems with sizes equal to the number of world states in the specified sub-model.

4.2 INTENTIONAL MODULE

The role of the intentional module is to handle the agent's objectives. We define an objective as a goal or as a chain of goals, which may be permanent or not, and that may possibly never be completed. For example, "maintain a high battery level" (which could be translated as "avoid having a low battery level") while constantly consuming energy, is an objective that cannot be completed once and for all. Objectives are modelled as *motivations*, as described in Section 3.

By managing motivations, the intentional module creates a motivation state vector (msv), which is a vector containing the current state of all motivations. Consequently, given a msv, it is possible to know all the rewarded motivation transitions originating from those current states, named *available-rmts*. This module is also responsible for keeping motivations up-to-date, depending on the world's evolution.

4.3 DELIBERATION MODULE

The deliberation module is the main module of our architecture. Its purpose is to choose autonomously the motivations to satisfy, and to find a *policy agenda* (i.e. a plan of policies) that will attempt to maximise the expected reward generated by the motivations for a given time horizon. Since there is no guarantee that the optimal solution for the problem can be decomposed into a series of optimal solutions for sub-problems, the computed solution is not guaranteed to be globally optimal. Nevertheless, this type of solution is considered *hierarchically optimal* (Kolobov, 2012; Dietterich, 2000).

In order to compute a suitable policy agenda, this module has to determine how the rewarded world transitions can interfere in the computed policies. A policy is computed in order to achieve a given rwt and ends when the wanted transition is executed. However, because the system operates under uncertainty, it may be possible that a policy aiming for $rwt1$ executes $rwt2$ instead ($rwt2$ is then said to "interfere" in the execution of $rwt1$). We made the choice of computing a policy that should aim for $rwt1$ while avoiding all other rewarded world transitions by giving to the MDP the transition $rwt1$ as well as all other interfering rwts, with nullified rewards in case these rewards were positive. Nevertheless, an unwanted rwt could still be executed. In this case, we want the policy to stop when executing it as well. This will be the case since the policies computed by the Operational Module terminate after executing any rwt received as input.

In a second phase, the deliberation module builds global states (gs), which represent the state of the entire system: $gs = (ws, msv)$. By executing a policy π_{rwt} until its end, we expect the world state ws and the motivation state vector msv to change as well. The deliberation process does not use the policies π_{rwt}, but rather their predictions ($Pr_\pi(ft|ws)$ and $C_\pi(ws, ft)$) made on those policies. These predictions are used to determine the transition between two global states, by computing the probability $pr(gs'|\pi, gs)$. The rmt executed from gs to gs' using π allows to compute $R(gs, \pi, gs')$, the obtained reward for transitioning to gs'. In this process $gs = (ws, msv)$, and $pr(gs'|\pi, gs)$ and $R(gs, \pi, gs')$ are respectively similar to states, transitions and rewards in an MDP but at a higher level.

The deliberation process can then predict the effects of the execution of a chain of policies on gs, as well as future *available-rwts* and the expected reward change. It builds a *policy tree* that gives, for each node corresponding to a global state, all possible gs' depending

on the policy executed from this node. Therefore, we are able to compute a local search in order to find the *policy agenda* maximising the average expected reward given by the motivations. The tree is developed until any of the following stopping criteria is reached: remaining time, maximum action depth, minimum probability threshold to reach a state. By using the calculated behaviour of policies, we compute the possible future global states up to a maximum reachable depth, their probability and the expected reward. Since we want to optimise the reward obtained for a given horizon of time, the main criteria for stopping the tree development is the time resource. Tree depth restriction avoids expensive computations of deep policy trees. The minimum-probability threshold needed for a node to be expanded avoids to search for very rare scenarios. If, however, a node is reached which was not expanded, a new tree will be computed, starting from this node.

Finally, a policy agenda can be seen as a policy of policies. Starting from a global state gs, the policy agenda provides the policy π_{rwt} to execute and all possible endings gs' for this policy execution, from which we can execute a next policy. So a policy agenda can be executed by executing the actions of the current policy as we do with a normal MDP, and by switching policies when the previous policy π_{rwt} stops.

4.4 ALGORITHM EXECUTION

This section shows what happens when the system initialises or when it receives a new request. A process can be divided in three phases: initialisation, deliberation and execution. We begin with the initialisation phase, computing all the necessary data, and then we alternate between phases of deliberation (by the deliberation module) and execution (through the supervisor) until the request is executed. In the absence of requests, the system has motivations that it is permanently trying to satisfy. The initialisation phase is only necessary when new motivations are introduced in the system. Figure 8 illustrates how the phases are ordered and executed.

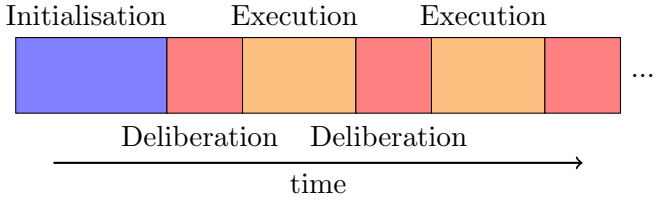


Figure 8: Algorithm execution. After the initialisation phase, the algorithm alternates between deliberation and execution phases. The colours correspond to the ones used in Figure 6.

- **Initialisation phase**:
  1. **Load the problem**: We begin by loading the world model (described using a PSO-like representation) into the **operational module** and then the motivations into the **intentional module**.
  2. **Pre-calculation of interferences in policies executing rewarded world transitions**: The **deliberation module** pre-computes the possible interferences between a policy π_{rwt} and the other rewarded world transitions.

3. **Sub-models creation**: For each selected rwt, the **deliberation module** asks the **operational module** to create the corresponding sub-model. The state space, the transition table and the cost function are computed for each sub-model using the general model given at start.
4. **Policies computation**: For each rwt and its sub-model, the **operational module** computes the optimal corresponding policy π_{rwt}.
5. **Policies behaviour computation**: For each rwt and its corresponding policy π_{rwt}, the **operational module** computes $Pr_\pi(ft|ws)$ (all possible outcomes and their probability) and $C_\pi(ws, ft)$ (the resource costs for all possible outcomes).
6. **Run initialisation**: The starting world state and the state of each motivation are set, constituting the starting *global state*.

- **Deliberation phase**: The deliberation phase consists in computing a *policy agenda* by the **deliberation module**. The process is divided in two steps:

  1. **Computation of the policy tree**: Starting from the current global state, we develop the policy tree in order to predict the outcome of the execution of the policies π_{rwt} corresponding to each *available-rwt*.
  2. **Calculation of the policy agenda**: Using the policy tree, we determine the optimal policy to execute from every node. Starting from the tree leaves, which correspond to future states in which *rmt* may or may not have been attained, we retro-propagate the rewards related to their corresponding action policies. We then select the optimal action policy to follow from each node, and repeat the selection process until we reach the root node. This gives us a new *policy agenda*.

  The computed *policy agenda* is then sent to the **supervisor module**.

- **Execution phase**: With the *policy agenda* given by the **deliberation module**, the **supervisor module** retrieves the policy corresponding to the *rmt* to execute from the starting global state. It executes this policy π_{rwt} until its end. After the execution of each action, the **intentional module** checks if the *available-rwt* have been executed, and gives a reward in this case. Once ended, the **supervisor module** searches for the new global state in the *policy agenda*, retrieves the new policy to execute, executes it and so on again until reaching to the end of the agenda.

4.5 COMPLEXITY OF THE ARCHITECTURE COMPUTATIONS

To evaluate the advantages of this architecture in terms of computational complexity, let us study its worst-case complexity for the different elements composing it. We begin by defining the symbols used in the complexity calculation, providing a short description for each of them.

- *nVar*: number of state variables,
- *nRules*: number of rules needed for the PSO-like definition of the world,
- *WS*: size of world state space (supposed to be finite),

- MSV: size of the global motivation state space (product of all motivation states),
- A: size of world action space,
- RE: number of problem resources,
- RMT: total number of rewarded motivation transitions,
- γ: MDP parameter for the convergence of Value-Iteration,
- WS_s: size of the largest sub-model state space,
- A_s: size of the largest sub-model action space,
- nFT: maximum number of final transitions for the rewarded motivation transitions,
- $nRMT$: upper bound of the number of *available-rwts* considering all motivation states,
- σ: parameter for stopping the policy tree development during deliberation. It corresponds to the combination of the parameters of units of time limit, depth limit and minimal probability threshold for expanding nodes. The combination of these parameters defines the policy tree's average depth.

Below, we give the worst-case complexity for all significant computations. The computations for which no complexity is given can be considered as less costly.

- **Initialisation**:
  - generation of the sub-models:
    $O(WS_s \times A_s \times nVar \times nRules) \times O(MSV \times RMT)$.
    The first factor describes the complexity of the generation of a sub-model. The second factor gives the number of sub-models calculated. However, in the average case, the complexity of this last part is $O(log(MSV) \times RMT)$, which is also valid for the computation of the policies and their behaviour computation, described below.
  - computation of the policies:
    $O({WS_s}^2 \times A_s) \times O(1/(1-\gamma)) \times O(MSV \times RMT)$.
    The complexity of the computation of all needed policies π_{rwt} is composed of three factors. The first corresponds to the worst-case complexity of a Value Iteration loop. The second factor is the number of Value Iteration loops needed to converge to the optimal policy. The last factor gives the number of policies π_{rwt} calculated.
  - policies' behaviour computation:
    $O({WS_s}^2 \times log(WS_s) \times nFT) \times O(MSV \times RMT \times RE)$.
    The complexity of the policies' behaviour computation is composed of two factors. The first one is the complexity of solving the equation system needed to calculate one prediction (either $Pr_\pi(ft|ws)$ or $C_\pi(ws, ft)$ for one resource only). The second factor gives the number of needed equation system resolutions, given the number of policies and the number of resources.

- **Deliberation**: The complexity for computing an policy tree is:
  $O(nRMT \times nFT \times RE) \times O((nRMT \times nFT)^{\sigma})$.
  The first complexity corresponds to the computation of the future global states for a given π_{rwt} and starting from gs. The second part is the number of nodes developed in the tree.

This analysis shows that all computations done by the architecture are done in a polynomial time, with the exception of the policy tree's computation. The use of *sub-models* reduces the complexity of computations, using WS_s instead of WS. Consequently, it is awaited that the sum of the multiple computations of sub-problems will be less costly than one computation of the entire problem. The impact of the complexity will be discussed in Section 5.7 based on a sample problem.

5. EVALUATION: ASSEMBLY EXAMPLE

We test our architecture on a fetch-and-carry scenario, with the constraints of: battery management, restricted access to specific areas, and damageable equipment. We voluntarily complexify the test case in order to demonstrate our system's ability to solve problems with a diverse range of constraints, by exploiting the problem's structure. In this example, a robot is placed in a factory, where it must build fans. For this, it has to assemble a frame piece, a motor and blades, using either a good machine (which flawlessly assembles pieces) or a bad one (which can fail and break pieces). All actions that the robot can do cost time and energy. The robot has to recharge its battery regularly, in order to avoid having a critical level of energy, risking to shut down. Every 3000 units of time, an area of the factory will be restricted for 600 units of time and will have to be avoided during this period. Finally, the chargers that allow the robot to recharge its battery can break down while charging, requiring some repair operations.

5.1 WORLD STATES

The world state is described by the following variables (see Fig. 9):

- **robot's position**: 15 possible positions, noted from "A" to "O".

- **doors' state**: the doors *door1* and *door2* can be either open or closed (their state can change when the robot deposit a fan on the conveyor belt).

- **robot's brakes**: *on* if the brakes of the robot are activated, *off* otherwise.

- **objects' location**: an object can be either in a robot hand, or stored in four different locations (i.e. storage1, storage2, or near the machines after a successful assembly), or unavailable if it has been consumed or destroyed. The available objects are: *frame*, *motor*, *blades*, *frame&motor* and *fan*.

- **chargers' status**: the status of the chargers, can be *on* if they are working, *problem* if they have a problem not identified yet, *deactivated* or *powerless*.

- **battery level**: a resource represented as a numerical value between 0 and 100, representing the percentage of energy left (100% corresponding to a fully charged battery, 0% to an empty one).

- **time**: a numerical value representing the elapsing time resource. It begins at 0 and is expressed in units of time.

Figure 9 illustrates the factory setting used in the example, together with the variables defining the world state. Grey cells, separated by lines (plain or dotted) represent the locations in which the robot may find itself. The robot image specifies its location. The doors are represented as coloured lines between cells: green ones are open, red ones are closed. Note that only *door1* and *door2* can be closed. If the robot's brakes are activated, the image of the robot is encircled. Each object is represented at its current location. The two storage places for objects are displayed in the upper-left and bottom-right corners. The slots of the results of the assemblies are displayed right next to the machines. The chargers are displayed on their location, and with a red lightning sign on them if their state is *off*, or a green one if they are *on*. The chargers' status is indicated in the bottom-right corner.

5.2 ACTIONS

The following actions are available to the robot:

- **move(direction)**: the robot can try to move from its current position to the adjacent ones. It has four directions of movement: up, down, left and right. This action is only possible if the robot's brakes are off, and if the target cell is accessible. The robot can only go through open doors. A movement has a 0.70 chance of success, 0.20 chance of staying in place and 0.10 chance to go to another accessible cell. If there is no other accessible cell, success chance is 0.80.

- **brake**: the robot can activate its brakes, or deactivate them if they are on. These actions always works. Having brakes on allows the robot to charge its battery, assemble objects, and increase the probability to take an object. However, the robot can only move when they are off.

- **recharge(amount)**: when the robot recharge its battery, it can do it by increments of 10% or 25% specified in the argument of this action. In order to use this action, the robot must be in a place with an active charger and have its brakes on. While charging, there is a risk of shutting down the chargers, setting their state to *problem*, in which they have to be repaired. Due to this, when charging 10% or 25% of the battery, the chance of success is 0.96 and 0.90, respectively.

- **take(object)**: the robot can try to take an object that is at the same location as itself. It must have at least one free hand. This action has 0.80 chance of success and 0.90 if the robot's brakes are activated. Failure has no effect the world state.

- **fetch(object)**: if one of the three basic objects (*frame*, *motor* or *blades*) is in the unavailable state, it can be retrieved by using this action. The robot must be on a storage location. After the action, the object's position is set to the storage location. This action always works.

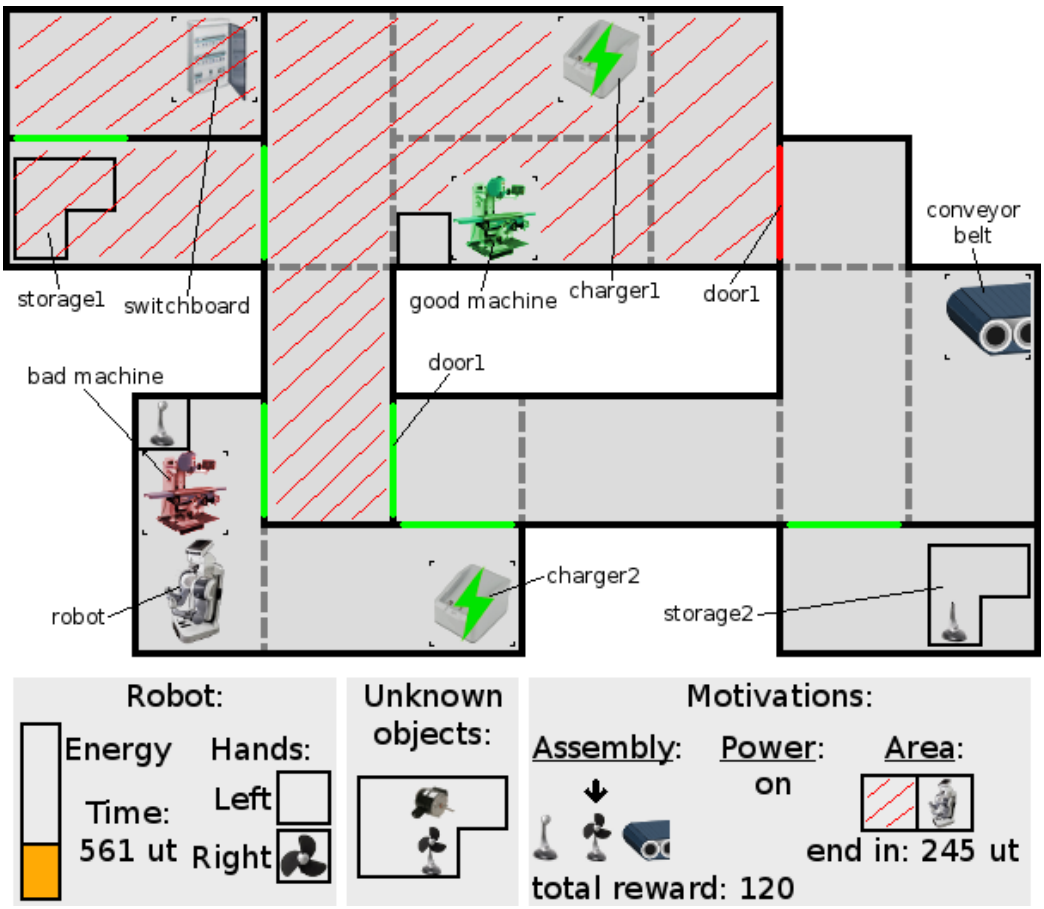


Figure 9: Graphic representation of the problem state. The battery level is represented by the gauge on the bottom-left, and the elapsing time value is shown on its right. In addition, the switchboard, the machines and the conveyor belt are displayed in the factory. The hatched area represents all the cells that can be restricted. The total time elapsed and the time before the restriction of the area ends are expressed in units of time, abbreviated "ut".

- **assemble**: this action consists in consuming two objects to make a new one. This action must be executed on a location with a machine, with the robot's brakes on, and with two objects in the robot hands. There are two working formulas: *frame* + *motor* gives *frame&motor* and *frame&motor* + *blades* gives *fan*. Success depends on the machine the robot uses. On the good machine, success rate is 1.00. On the bad machine, success rate is 0.60. In case of success, the first two objects are set to *unavailable*, and the resulting object's location is set to the result slot of the used machine. On failure, there is 0.20 chance of breaking one of the two objects, 0.10 chance to break both, and 0.50 chance that nothing happens.

- **deposit(fan)**: the robot deposits a fan on the conveyor belt. The robot must be in the same place as the conveyor belt, have its brakes on and have a fan in its hands.

This action always succeeds. After the action, the fan's state is set to *unavailable*. Moreover, the doors *door1* and *door2* have each 0.50 chance to switch state.

- **identifyProblem**: discovers the cause of the chargers' problem. The robot must be on one of the chargers, and the chargers' state must be *problem*. This action identifies the chargers' status with 0.50 chance as *deactivated* and *powerless* at 0.50.
- **putPowerOn**: put the chargers' power back on. The robot must be in front of the switchboard and the chargers' status must be *powerless*. This action has a success rate of 1.00 and change the chargers' status to *deactivated*.
- **activateChargers**: put the chargers in operation. The robot must be in the same location as one of the chargers, the chargers' status must be *deactivated*. This action always works and sets the chargers' status to *on*.

5.3 ACTION COSTS

As mentioned previously, each action has a cost in terms of resources: time and energy. These costs are detailed in Table 1, depending on the outcome of actions. Time cost is expressed in units of time.

5.4 MULTIPLE OBJECTIVES

In order to solve the *assembly problem*, the robot has to take into account four different simultaneous objectives. The first objective is the robot's survival. The robot must avoid loosing all of its energy. If this happens, the robot stops, and so does the experiment. The second objective is the assembly task: the robot has to produce fans and deposit them on the conveyor belt. The third one consists in maintaining the power of the chargers. If power is lost, the robot has to switch it on again. The fourth objective concerns the restricted area. When the restriction is active, the robot must avoid the restricted area. Below, we show how objectives are defined.

- **Survival**: We divide the robot's battery gauge into four intervals: $[0-10\%]$ labelled *critical*, $[10-40\%]$ labelled *low*, $[40-70\%]$ labelled *high*, and $[70-100\%]$ labelled *full*. The objective is to avoid the critical interval, and to favour the full interval. It defines a rmt consisting in recharging the battery once. The objective gives a positive reward each time a recharge action is executed as well as when the energy value changes from an interval to higher one. Similarly, the objective gives a negative reward when the energy value changes from an interval to a lower one. To avoid charging forever, this *rmt* is available only in the three first intervals. Finally, since the *critical* interval must be absolutely avoided, we give a penalty every 10 units of time when the battery state is in this interval.
- **Assembly**: This objective consists in producing fans and depositing them on the conveyor belt. To do this, this objective gives a chain of three rewarded motivation transitions, which, when the last one has been executed, loops back to the first one. For the first *rmt*, the robot has to machine a *frame+motor* object. For the second *rmt*, it has to build a *fan*. For the last one, it has to deposit the *fan* on the conveyor

action	**resource**	**success**	**failure**
move	time	+7	+14
	energy	−0.3%	−0.5%
brake	time	+2	
	energy	−0.1%	
recharge	time	+20 / +40	+1
	energy	+10% / +25%	−0.1%
take	time	+8	+11
	energy	−0.2%	−0.4%
fetch	time	+6	
	energy	−0.6%	
assemble	time	+20	+28
	energy	−0.3%	−0.4%
deposit	time	+4	
	energy	−0.2%	
identifyProblem	time	+10	
	energy	−0.3%	
putPowerOn	time	+4	
	energy	−0.3%	
activateCharges	time	+2	
	energy	−0.1%	

Table 1: Table of action costs, depending on their outcome. The exact values are irrelevant

belt. All of these *rmt* give a reward. Finally, we have to take into account the possibility of breaking the *frame+motor* or the *fan* objects. If one of the two objects is accidentally destroyed, the objective rolls back to the first one and gives a penalty.

- **Maintain power**: In order to recharge its battery, the robot has to maintain the power of the chargers. This objective gives a penalty when the chargers switch their state to *problem* while charging. This state contains a rewarded motivation transition that consists in switching the power on by repairing the chargers, which gives a reward when executed.

- **Avoid restricted area**: Every 3000 units of time, the area composed of the upper corridor and the double room in the upper-left corner is restricted for 600 units of time. While restricted, this objective gives a penalty when the robot enters the area, as well as every 10 units of time when it remains inside it. This implies that, while the area is restricted, the robot should avoid:

  – taking objects from *storage1*,

  – using the *good machine*,

  – recharging at *charger1*.

  Also, if the chargers become *powerless*, the robot will not be able to repair them without getting a penalty.

The four objectives and their respective states are displayed on the graphical representation, in the bottom-right corner of Figure 9 (except for the *survival* objective). The four intervals of the *survival* objective are represented by the colour of the current value on the battery gauge, in the robot section, in the bottom-left corner. For the *assembly* objective, three icons, representing the objects *frame+motor* and *fan* and the conveyor belt, are displayed representing each steps. A pointer indicates the next step to achieve. The *maintain power* objective is represented with the chargers' status. The objective relating to the *restricted area* is displayed depending on the area status. When the area is not restricted, the remaining time before restriction is indicated. When restricted, it depicts if the robot is in the restricted area or not and the time before the restriction ends.

5.5 OBJECTIVES AS MOTIVATIONS

Based on the descriptions of the four objectives given in Section 5.4, we implemented four corresponding motivations for our deliberative system. Each motivation has its own motivation states. The deliberative system uses a vector describing the current joint state of the motivations. The chosen reward values on the rewarded motivation transitions are irrelevant.

The motivation presented in Figure 10 illustrates the *Survival* objective. Motivation states **critical**, **low**, **high** and **full** correspond respectively to the intervals $[0\% - 10\%]$, $[10\% - 40\%]$, $[40\% - 70\%]$ and $[70\% - 100\%]$. The rewarded motivation transitions between those states change the state of this motivation depending on the battery level, and give rewards when the battery level is increased, and penalties otherwise. On the right of states **critical**, **low** and **high**, the self-loops correspond to the *rmt* of recharging the battery in

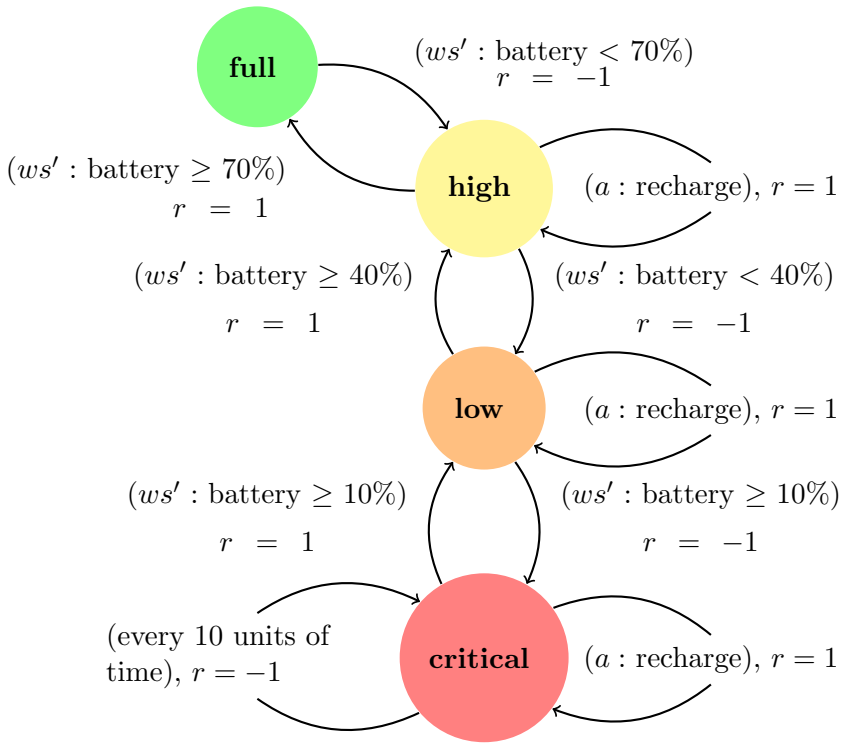


Figure 10: Scheme of the *Survival* motivation, defined in terms of battery charge levels.

these states (we chose to make recharging unavailable in the motivation state **full**, to allow the agent to concentrate on other tasks when there is enough charge). Recharging can be executed using *charger1* or *charger2*. The last *rmt* is a self-loop on state **critical** and gives a penalty for every 10 units of time spent in this state.

Figure 11 illustrates the *Assembly* objective, modelled in its motivation form. The motivation is composed of three motivation states: **step1**, **step2** and **step3**, that correspond to the three steps of the assembly objective. The first one corresponds to the state in which a *frame&motor* object must be created with the assembly action, either with the good or the bad machine. The corresponding rewarded motivation transition sets the state of this motivation to **step2** and gives a reward. The motivation state **step2** is the state where the *fan* must be assembled. The *rmt* starting from **step2** and leading to **step3** is triggered when the assembly action succeeds. On the other hand, if the assembly action fails, destroying the *frame&motor* previously created, it executes the *rmt* leading to **step1**, where the object must be assembled again. Finally, the state **step3** is the state where the **fan** must be put on the conveyor belt. Once the *fan* is on the conveyor belt, the rewarded motivation transition is activated, leading the agent to obtain a reward and setting the motivation state back to **step1**.

The third objective is *Maintain power* and is represented in Figure 12. It is a simple motivation where the motivation state is **activated** when the chargers' status is *on* and **deactivated** in all the other cases (*problem*, *deactivated* or *powerless*). The *rmt* from **ac-**

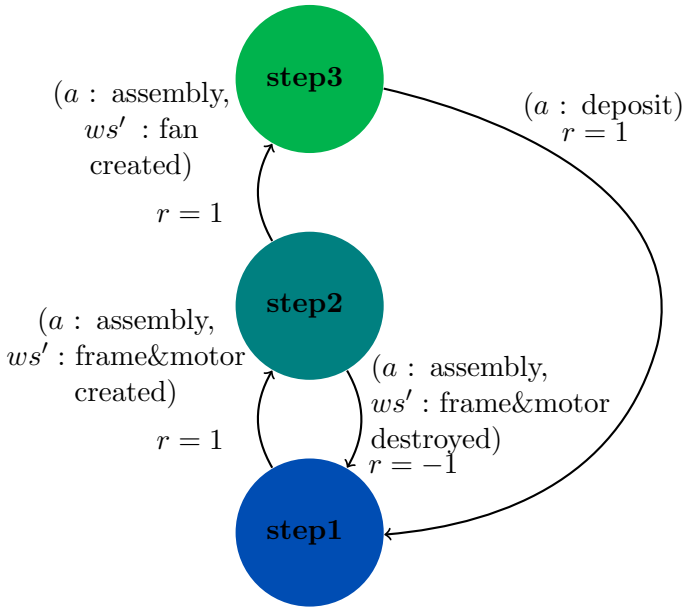


Figure 11: Scheme of the *Assembly* motivation.

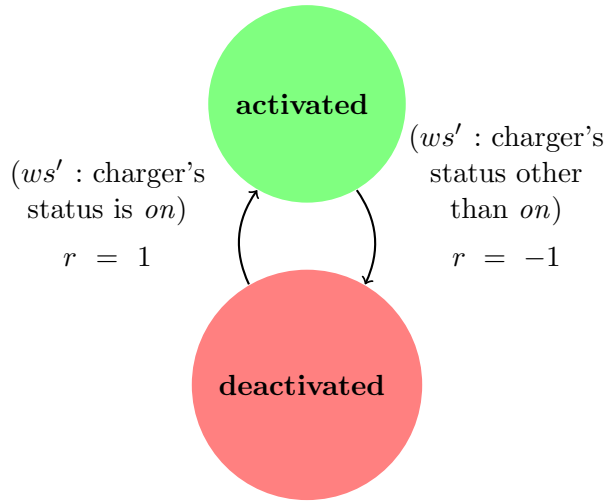


Figure 12: Scheme of the *Maintain power* motivation

tivated to **deactivated** becomes activated when the chargers lose power after a *recharge* action, and inflicts a penalty. The *rmt* on the left is activated when the chargers' status becomes *on* again, which can happen after using the actions *activateChargers* and *putPowerOn*. On activation, the robot gets a reward and the state of this motivation is set to **activated**.

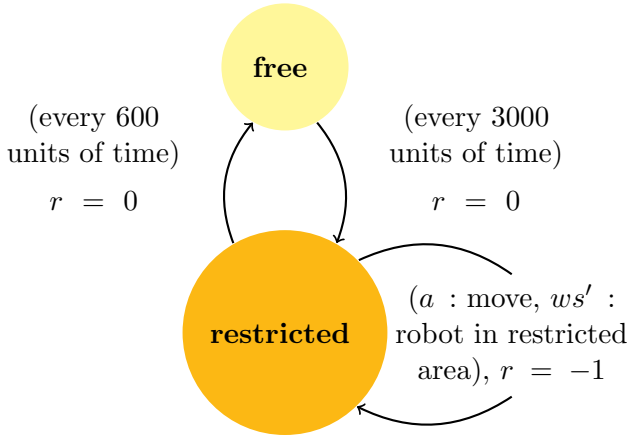


Figure 13: Scheme of the *Avoid restricted area* motivation

The last objective described is the *Restricted Area*, represented in Figure 13. The state of this motivation is dependent only on the evolution of the time resource and changes as it elapses: the state **free** represents the situation where all the factory is in free access, and **restricted** where the area (visible on Figure 9) is restricted. The motivation alternates between the states **free** (3000 units of time) and **restricted** (600 units of time). Note that the two rewarded motivation transitions making the state change possible are set with a null reward, since this state change does not have any positive or negative aspect. In the **restricted** state, the area is restricted and the robot receives a penalty each time it goes into the area. Consequently, when the current state is **restricted**, some *rmt* of the other motivations can be executed by incurring a penalty. This is the case for the rewarded motivation transitions of recharge, assembly and getting power back on, from the respective motivations *Survival*, *Assembly* and *Maintain power*.

5.6 EXPERIMENTAL RESULTS

We tested our deliberative system experimentally using the described problem. In order to demonstrate the utility of the deliberation module, we compared the results obtained in the presence and the absence of the policy tree generated by the deliberation module. For each tested configuration, we ran the algorithm 20 times. The starting world state for each run was:

- the robot starts near the conveyor belt with brakes off,
- the two doors are open,
- all items are unavailable,
- both robot's hands and the storage locations are empty,
- the chargers are working,

- the battery begins with 55% energy level, and time at 0 units of time.

The starting motivations states were:

- the *Energy managing* objective starts in *high* state,
- the *Assembly* objective is in its initial state, waiting for the *frame&motor* object to be assembled,
- the *Area* is restricted for 600 units of time,
- the objective *maintainPower* is in the state where chargers' status is *On*.

The experiment lasts until the time resource is superior to 21600 units of time.

In its current implementation, the problem presented here has around 2.4×10^7 world states, 24 possible actions (without counting resource variables) and 48 possible joint motivation states.

The process can be divided into three phases: initialisation, deliberation computation and execution (see Section 4.4). The initialisation, which generates all needed *sub-models*, policies π_{rwt} and policies' behaviour for our example problem, is shared by all tests performed with our architecture. It takes 220 seconds on one 3.4 GHz processor and consumes less than 1 GB of RAM storage.

Our architecture can be configured by defining the criteria for stopping the development of the policy tree computed by the **deliberation module**. For our tests, we began by using the following parameters: the tree stops at the horizon of 21600 units of time. The deliberation computation (computed online) takes in total approximately 700 MB of RAM storage. In the given time frame, the deliberation computation is called around 90 times, for a total computation time of 490 seconds. The deliberation generally takes between 1 and 10 seconds. The search space had an average size of 7500 states (this data is given by the number of nodes explored in the policy trees computed to create the policy agenda). From each global state in the policy trees, the average number of *available-rwts* was 2.8 and the average number of possible future global state for these *rmt* was 2.2 and the average policy tree depth was 4.6.

Since we did not find any other comparable algorithm or architecture that could handle our problem, we evaluate our algorithm by analysing the impact of the different parameters on its execution.

To verify the validity of our deliberation system, we compare the algorithm with itself, but with a limited deliberation. For this test, we compare two runs, one with a nominal configuration of 180 units of time, the other with a horizon of 1 unit of time. With the limited configuration, the second run only chooses immediate reward and does not take into consideration the long term reward. Figure 14 shows the reward accumulated by all motivations depending on the elapsed time resource. In this test, the algorithm with nominal parameters obtains more reward than the one with limited deliberation time (4200 against 2600 total reward). The algorithm with a nominal configuration also finished more than 40 assembly cycles (starting from *step1* and going back to *step1* after dropping a fan on the conveyor belt), while the algorithm with limited deliberation time finished only 30 cycles.

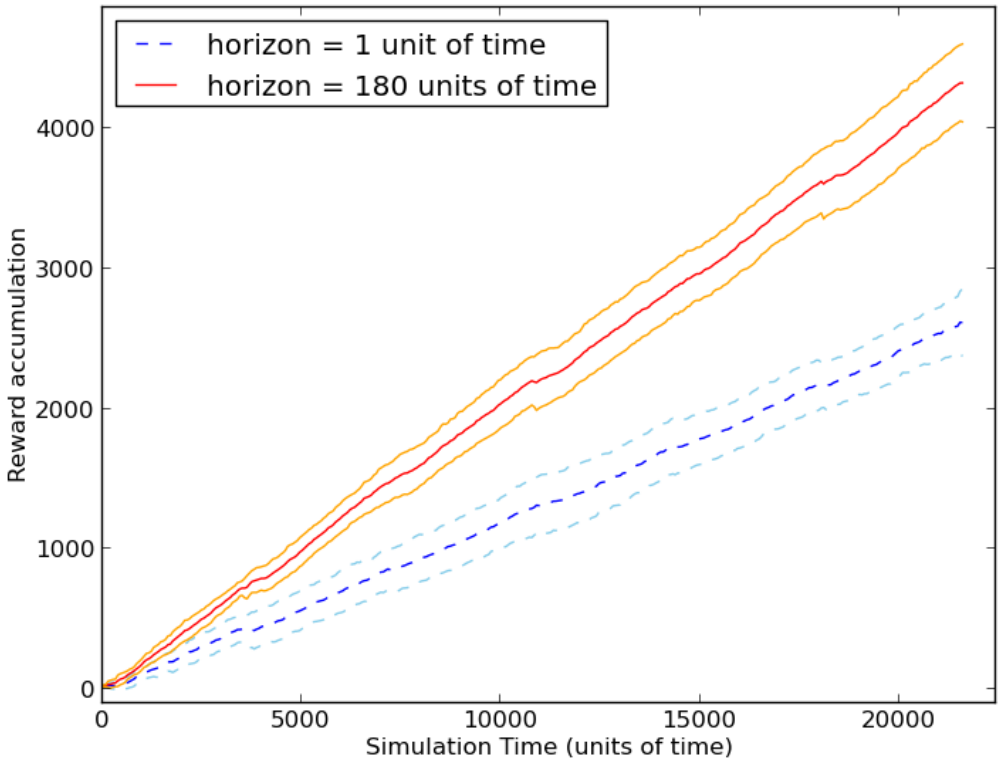


Figure 14: Test run between a nominal configuration (red plain curve) and limited deliberation (blue dashed curve). Each curve as its standard deviation indicated with the upper and lower bounds

Also, we want to illustrate the impact of the three parameters, time horizon (expressed in units of time), the depth limitation and the minimal probability of reaching a node in order to expand it, that determine when the policy tree development stops. We analyse the following configurations:

- **nominal configuration**: horizon of 180 units of time, maximum tree depth set to 5 and minimal probability for expanding a node is 0.005.
- **non-restricted configuration**: horizon of 180 units of time, maximum tree depth is now set to 7 and minimal probability for expanding a node is now 0.001.

We compare the differences in terms of efficiency (reward accumulation for the experience) and the time needed to compute the policy trees with the two configurations. Figure 15 illustrates the results of our test. One can see that if the reward accumulated by the non-limited configuration is slightly higher than for the standard one, the time needed to compute the actions trees in this configuration is about 10 times higher than in the nominal

case. The difference in computation time is caused by the maximum tree depth and the minimal probability threshold parameters, which change the average depth from 4.6 to 6.1. For the same reasons, the RAM consumption changed from less than 2 GB to more than 4 GB. Hence, since the difference between the reward accumulated by the two configurations remains small, this confirms that use the parameters that limit the tree depth and that only develop nodes above a minimal probability help to reduce the computation time without reducing significantly the quality of the results.

5.7 Discussion

The proposed architecture was run on a 3.4 GHz CPU, consuming less than 2 GB of RAM to execute a problem with a non-trivial state space. The problem is modelled with probabilistic outcomes of actions, and with the use of resources (used to represent battery charge and time). This is a class of problems for which we did not find any existing suitable probabilistic planners. Moreover, we chose to work on looping objectives to explore tasks such as survival.

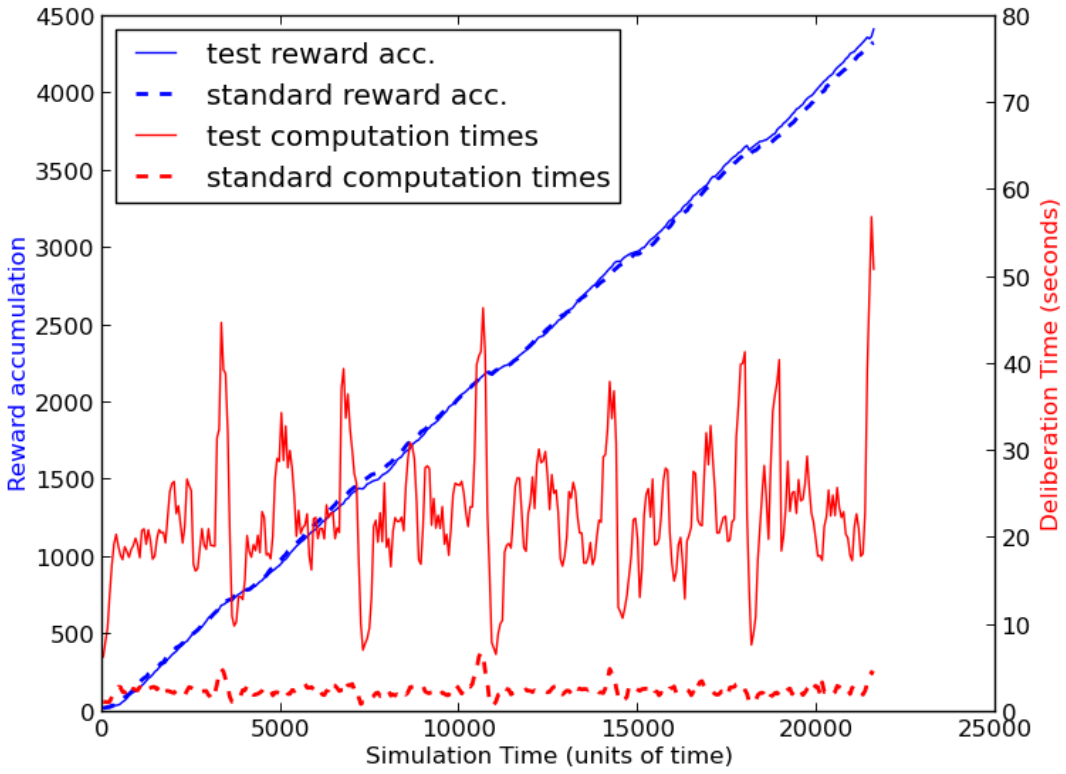


Figure 15: Comparison between nominal configuration (dashed curve) and non-limited configuration (plain curve).

The defined world had about 2.4×10^7 world states, without counting resource variables. This makes the generation of the corresponding transition table infeasible in practice, due to the memory consumption needed for building the matrix of size $WS \times WS \times A$. Instead of building the complete table, we represent the world model with PSO-like rules. These rules allow to build sub-models for computing the policies used to reach the given rwt. These sub-models are partial definitions of the complete model and, in our example, have a maximum size of 4.8×10^4 states (and most of them under 5000 states), with a limited number of actions. This reduced size of sub-models allow to compute their transition table, and to compute policies in a reasonable time.

Thus, this architecture takes advantage of the way in which a problem can be decomposed, as explained in Section 4.5. The decomposition of the problem centres on:

- How many world variables each action depends on, and how many variables the action acts on. The less variables the actions depend on, the easier it is to decompose the model.

- The number of different actions needed to execute a rmt. If the number of actions is low, the corresponding sub-model will be small and the corresponding computations fast.

- The way the objectives can be decomposed into their motivation form. The definition of the rewarded motivation transitions (and thus rwt)can have a heavy influence on the number and the nature of actions needed to execute the corresponding policies π_{rwt}, and thus on the architecture performance.

6. CONCLUSION

In this paper, we introduced a system for managing multiple concurrent and permanent objectives, and generating plans in order to satisfy them, that we call a *deliberative system*. The system performs probabilistic reasoning while dealing with multiple objectives, and can perform judgements based on resources. It employs a novel way for structuring objectives in the form of finite state machines, which we call *motivations*.

The deliberative system has a modular architecture, which separates the planner from the goal-manager entity, allowing for an easy integration into an existing robotic cognitive architecture. Moreover, the motivation-based deliberative process renders the architecture capable of initiative and not only reactive to external requests.

The system was evaluated on a problem instance with complexity of the search space ranging in the 2.4×10^7 states, for which it was able to provide sub-optimal results in approximately 5 minutes on a 3.4 GHz processor.

Due to the hierarchical-planning method it employs, the system provides no guarantee on solution optimality. However, this way of solving the problem is termed *hierarchically optimal* (Kolobov, 2012; Dietterich, 2000). Therefore, considering how the architecture lowers the problem complexity by decomposing the problem, the algorithm can be employed in practice. This statement is supported by the results obtained for the presented sample problem.

A proper comparison of the system's performance with those of other systems could not be performed, as (to our knowledge) there are currently no planners in its category, capable of solving probabilistic multi-goal planning problems with resources.

Future work will involve an integration with the RDDL problem description language (which is currently only partial), in order to replace the custom version of STRIPS problem descriptor language that is currently used. An integration of the whole motivational architecture into a robotic cognitive architecture is also underway in the context of the RoboErgoSum project.

An open problem remains as to the grounding of the symbolic concepts used in defining the motivations. We plan to do it by letting the robot to automatically learn and construct representations of objects and their potential use (actions), from a minimal initial set, using the concept of *affordance*. With these auto-generated concepts, we hope to empower the robot to autonomously identify, select and pursue objectives.

Acknowledgments This work has been funded by French Agence Nationale de la Recherche *ROBOERGOSUM*[1] project under reference ANR-12-CORD-0030.

References

Alami, R., Chatila, R., Fleury, S., Ghallab, M., & Ingrand, F. (1998). An architecture for autonomy. *The International Journal of Robotics Research*, *17*(4), 315–337.

Boutilier, C., Dean, T., & Hanks, S. (1999). Decision-theoretic planning: Structural assumptions and computational leverage. *Journal of Artificial Intelligence Research*, *11*(1), 94.

Bresina, J., Dearden, R., Meuleau, N., Ramakrishnan, S., Smith, D., et al. (2002). Planning under continuous time and resource uncertainty: A challenge for ai. In *Proceedings of the Eighteenth conference on Uncertainty in artificial intelligence*, pp. 77–84. Morgan Kaufmann Publishers Inc.

Coddington, A. (2007). Motivations as a meta-level component for constraining goal generation. In *In Proceedings of the First International Workshop on Metareasoning in Agent-Based Systems*, pp. 16–30.

Coddington, A. M., & Luck, M. (2004). A motivation-based planning and execution framework. *International Journal on Artificial Intelligence Tools*, *13*(01), 5–25.

Dietterich, T. G. (2000). Hierarchical reinforcement learning with the maxq value function decomposition. *J. Artif. Intell. Res.(JAIR)*, *13*, 227–303.

Gat, E. (1992). Integrating planning and reacting in a heterogeneous asynchronous architecture for controlling real-world mobile robots. In *AAAi*, Vol. 1992, pp. 809–815.

Georgeff, M. P., & Lansky, A. L. (1987). Reactive reasoning and planning.. In *AAAI*, Vol. 87, pp. 677–682.

Ghallab, M., & Laruelle, H. (1994). Representation and control in ixtet, a temporal planner.. In *AIPS*, Vol. 1994, pp. 61–67.

1. roboergosum.isir.upmc.fr

Guestrin, C., Hauskrecht, M., & Kveton, B. (2006). Solving factored mdps with hybrid state and action variables. *Journal Of Artificial Intelligence Research*.

Ingrand, F., & Ghallab, M. (2014). Deliberation for autonomous robots: A survey. *Artificial Intelligence*, 1–40.

Kolobov, A. (2012). Planning with markov decision processes: An ai perspective. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, *6*(1), 1–210.

Lemai, S., & Ingrand, F. (2004). Interleaving temporal planning and execution in robotics domains. In *AAAI*, Vol. 4, pp. 617–622.

Mausam, & Weld, D. S. (2008). Planning with durative actions in stochastic domains.. *J. Artif. Intell. Res.(JAIR)*, *31*, 33–82.

Meuleau, N., Benazera, E., Brafman, R. I., Hansen, E. A., & Mausam, M. (2009). A heuristic search approach to planning with continuous resources in stochastic domains. *Journal of Artificial Intelligence Research*, *34*(1), 27.

Muscettola, N., Nayak, P. P., Pell, B., & Williams, B. C. (1998). Remote agent: To boldly go where no ai system has gone before. *Artificial Intelligence*, *103*(1), 5–47.

Nau, D. S., Au, T.-C., Ilghami, O., Kuter, U., Murdock, J. W., Wu, D., & Yaman, F. (2003). Shop2: An htn planning system. *J. Artif. Intell. Res.(JAIR)*, *20*, 379–404.

Nesnas, I. A., Wright, A., Bajracharya, M., Simmons, R., & Estlin, T. (2003). Claraty and challenges of developing interoperable robotic software. In *Intelligent Robots and Systems, 2003.(IROS 2003). Proceedings. 2003 IEEE/RSJ International Conference on*, Vol. 3, pp. 2428–2435. IEEE.

Pollack, M. E., & Horty, J. F. (1999). There's more to life than making plans: plan management in dynamic, multiagent environments. *AI Magazine*, *20*(4), 71.

Rabideau, G., Knight, R., Chien, S., Fukunaga, A., & Govindjee, A. (1999). Iterative repair planning for spacecraft operations using the aspen system. In *Artificial Intelligence, Robotics and Automation in Space*, Vol. 440, p. 99.

Sanner, S. (2011). Relational dynamic influence diagram language (rddl): Language description (2010). *URL http://users. cecs. anu. edu. au/˜ ssanner/IPPC_2011/RDDL. pdf. Citado na pág*, *55*, 59–79.

Sutton, R. S., Precup, D., & Singh, S. (1999). Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, *112*(1), 181–211.

Glossary

MDP Markovian Decision Process. Probabilistic decision model in which actions can have several possible outcomes.

HTN Hierarchical Task Network. In deterministic decision model, a method allowing to use partial plans akin to actions in future planning.

PSO Probabilistic STRIPS Operator. Allows to describe an action in order to be used in a probabilistic decision model, like MDP.

ws World state. Usually abridged s, the robot's and world's state used by MDPs. Abridged ws to avoid confusion with motivation states and global states.

ms Motivation state. The state of the automaton of one motivation.

rmt Rewarded motivation transition. A transition between to motivation states, triggered by a specific world transition. Gives a reward when triggered. Noted $(ms, (ws, a, ws'), ms'), r$.

ft Final transition. Specific world transition (ws, a, ws') used by our customized MDP. A policy created with this ft terminates when this transition is executed.

rwt Rewarded world transition. Extracted from a rmt and noted $(ws, a, ws'), r$. A world transition used as a ft in order to create a π_{rwt}.

π_{rwt} Policy aiming for rwt. A policy that have been computed in order to reach the final transition corresponding to rwt.

msv Motivation state vector. A vector containing the state value of the motivations.

gs Global state. The global state of the system, defined (ws, msv).