# Executing Concurrent Actions with Multiple Markov Decision Processes

Elva Corona-Xelhuantzi, Eduardo F. Morales and Enrique Sucar
National Institute of Astrophysics, Optics and Electronics,
Department of Computer Science, Tonantzintla, Puebla, 72840, MEXICO

*Abstract*— **Markov decision processes (MDPs) have become a standard method for planning under uncertainty, however they usually assume a sequential process, so a single action is executed at each time step. In some applications, as in robotics, it is required to execute several actions concurrently. For this we propose a framework based on a functional decomposition of the problem into several sub-problems, each represented as a $subMDP$. Each $subMDP$ is solved independently and their policies are combined to obtain a global solution, such that the actions of each $subMDP$ can be executed concurrently. As we combine the local policies, conflicts between them can arise. We define two kinds of conflicts, *resource* and *behavior conflicts*, and propose solutions for both. Resource conflicts are solved off–line via a two-phase process which guarantees a near–optimal global policy. Behavior conflicts are solved on–line based on a set of restrictions specified by the user. If there are no restrictions, all the actions are executed concurrently; otherwise, an arbiter selects the action(s) with higher expected utility. We present experimental results in two cases: (i) a simulated robot navigation problem, with *resource conflicts*, and (ii) a simulated robot in a message delivery task, with *behavior conflicts*.**

*Index Terms* — **Markov Decision Processes, Concurrent Actions, Service Robots**

## I. Introduction

Automated planning is the computational process of generating a course of action (*plan*) for an agent (e.g. a robot) to execute. A planner takes as input a description of the world (state), and outputs a sequence of actions (a "plan"), or a function from world-state to actions (a "policy"). Markov decision processes (MDPs) are a standard model for this kind of systems, that exhibit probabilistic and nondeterministic behaviors. A dynamic process can be modeled as an MDP, which evolves through stages, in each stage chooses one of several actions, and the system stochastically evolves to a new state based on the current state and the chosen action. The solution to an MDP determines a policy which specifies the action to be selected at each time step, such that a certain objective function is maximized (usually the expected accumulated reward in the future). However, MDPs usually assume sequential processes, so a single action is executed at each time step. In some applications, as in robotics, it is required to concurrently execute several actions, for instance, navigating while interacting with a user. A possible solution is to define each action as a *set of actions*, but this increases significantly the size of the model, and it makes more complex its specification. As an alternative solution, we propose a framework based on a functional decomposition of the problem, which is partitioned into several simpler sub-problems, each represented as a $subMDP$. Each $subMDP$ is solved independently and their policies are combined to obtain a global solution, such that the actions of each $subMDP$ can be executed concurrently. An additional advantage of this decomposition is that the sum of the size of the individual models is usually smaller that the complete MDP, and thus the space and time complexities for solving the problem are reduced. Many problems can be naturally divided into functional modules; for instance, in robotics, we can define modules for navigation, localization, manipulation, human–robot interaction, etc. This makes the proposed decomposition a natural framework for this type of applications and facilitates the definition or learning of the models.

The functional decomposition assumes that the sub processes are relatively independent, however, as we combine the local policies, conflicts between them can arise. We define two kinds of conflicts, *resource* and *behavior conflicts*, and propose solutions for both. Resource conflicts occur when two actions require the same physical resource (i.e., to control the wheels of a robot) and can not be executed concurrently. This type of conflicts are solved off–line via a two-phase process. In the first phase we obtain an optimal policy for each $subMDP$ using a standard method. An initial global policy is obtained by combining the local policies, such that if there is a conflict between the actions selected by each $subMDP$ for certain state, the one with maximum value is considered, and the state is marked as a *conflict* state. This initial solution is improved in a second phase using policy iteration [14], taking the previous policy as its initial policy, and considering only the states marked as conflicts; with this consideration the time complexity is drastically reduced, and a near-optimal global policy is obtained.

Behavior conflicts arise in situations in which it is possible to execute two (or more) actions at the same time, but it is not desirable given the application. For example, it is not desirable for a mobile robot to be navigating and handing out an object to a person at the same time (this situation is also difficult for a person). Behavior conflicts are solved on–line based on a set of restrictions specified by the user. If there are no restrictions, all the actions are executed concurrently; otherwise, an arbiter selects the action(s) with higher expected utility. In this way the selected action in case of conflicts is not always the same, as it depends on the current state; in this case we can not guarantee a global optimum.

The proposed framework is being applied to a service robot which delivers messages and objects between persons in an indoor environment. We present experimental results in two cases: (i) a simulated robot navigation problem, with *resource conflicts*, and (ii) a simulated robot in a message delivery task, with *behavior conflicts*.

The rest of the paper is structured as follows. We first present a brief overview of MDPs and related work in abstraction and decomposition. Then we introduce the proposed architecture and describe both techniques for solving conflicts. Finally, we describe the experimental evaluations and conclude with directions for future work.

## II. Markov Decision Processes

A discrete Markov decision process [14] models a dynamic process, the environment is observed in each step by the agent, it selects an action, performs the action selected (which modifies the environment), and the agent receives a reward depending of the change. An MDP is formally defined as follows:

- $S$ is a finite set of states $s_1, s_2, s_3, ..., s_n$.
- $A$ is a finite set of actions $a_1, a_2, a_3, ..., a_m$.
- $\Phi(s, a, s^{'})$ is the probability of moving to state $s^{'}$ when action $a$ is executed in state $s$.
- $R$ is a reward function. $R(a, s)$ specifies the immediate reward of taking action $a$ in state $s$.

A solution to an MDP is a policy, $\pi$, that specifies the action to be executed by the agent at every state. The optimal policy ($\pi^*$) is usually the one that maximizes the expected accumulated reward, $V^*$, such that it satisfies Bellman's equation [2]:

$$V_{\pi}(s) = max_a\{R(s,a) + \gamma \sum_{s^{'} \in S} \Phi(s,a,s^{'})V_{\pi}(s^{'})\} \qquad (1)$$

where $\gamma$ is a discount factor. Two popular methods for solving this equation and finding an optimal policy are: (i) *Policy Iteration* and (ii) *Value Iteration* [14].

The main drawback of MDPs is due to the *curse of dimensionality*. Although the temporal complexity of traditional solution methods is polynomial on size of the *state-action* space, this can be very large, growing exponentially in the number of states and actions. In its basic form, policy and value iteration require an explicit representation of states and actions and need to explore the entire state space during each iteration. In particular, if multiple concurrent actions are allowed, this will imply a further increase in complexity, as all actions combinations need to be considered.

To deal with the complexity problem there are three main approaches: *factorization,* in which the state space is represented in a factored form (e.g. [3], [8]); *decomposition*, that divides the global problem into smaller problems that are solved independently and their solutions are combined (e.g. [12], [10]); and *abstraction*, that creates an abstract model where states with similar features are grouped together(e.g. [13], [5], [9], [11], [4]). Next we describe each alternative.

### A. Factored MDPs

Factored MDPs address the complexity problem via compactly specifying the model of the MDP in factored form; the state space (**S**) is modeled by a set of variables $X = X_1, .., X_n$, and the actions are described as having an effect on specific variables under certain conditions, implicitly inducing the transition function represented as a *dynamic Bayesian network* (DBN) [7]. For instance, [3] uses a $two-slide$ *temporal Bayesian network* (TBN) [16] to represent the dependencies between variables before and after the occurrence of an action, and a structured decision tree to represent the transition and reward functions. The solution algorithm retains the basic steps of policy iteration, but it exploits the independencies reflected in the TBN. SPUDD [8] uses algebraic decision diagrams (ADDs) [1] to represent the transition and value functions, and based on this representation it uses very efficient techniques for ADDs to implement value iteration. Although factored representations allow to solve quite *large* MDPs, they do not address directly the problem of concurrent actions.

### B. Abstraction

Under the abstraction approach, a group of states in the original MDP is mapped to a single abstract state. For this, the group of states must be *equivalent*, or at least share the same local behavior. For instance, [9] is an approach that discovers a set of state variables that are irrelevant for a policy, finds the states where this set of variables is irrelevant, and encapsulates them into one state using temporal abstraction. Hierarchical abstract machines (HAMs) [13] consist of non−deterministic finite state machines whose transition may invoke other lower level machines. These approaches also do not solve the issue of concurrent actions.

### C. Decomposition

Decomposition is based on the old principle of *divide and conquer*. In [12] the problem is divided into smaller tasks. In an off–line phase, the value function and optimal policy for the MDP associated to each subtask is computed. In an on–line phase, these value functions are used within a heuristic search procedure to assign resources to each task, and depending on the resources allocated, the action to execute is selected. In [10] the state space is divided into physical regions (office, corridor, room, etc.), and each one is solved as an MDP. The regions have communication with their neighbors through an state (initial or goal) called *intersection*. Then a directed graph is built, where each intersection is a vertex and the regions are the arcs. Each arc is associated to the value function of its respective MDP, and to get a solution the shortest path is found. In [5], the MDP is divided into several subtasks, identifying a subset of state variables which are *relevant* for each subtask. It then defines a value function and policy using only these relevant variables for each task.

Most decomposition approaches make a *serial* decomposition; that is, the problem is partitioned in subtasks that are executed sequentially. These techniques can not execute concurrent actions. There are few previous work on *parallel*

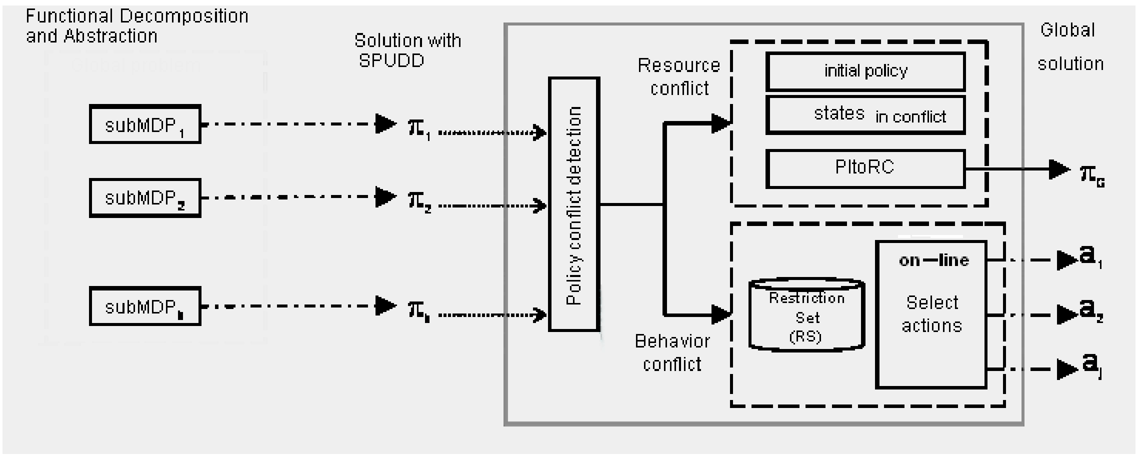

Fig. 1. The general architecture is composed by four phases: (a) functional decomposition; (b) definition and solution of the subMDPs; (c) detection of conflicts between local solutions and combination; and (d) execution of the global solution.

*decomposition* (e.g. [12], [15]), where the subtasks are executed concurrently, so in principle they can execute several actions simultaneously. Elinas et al. [6] propose an approach for coordinating a service robot based on concurrent MDPs, assuming the subtasks are independent and that there are no conflicts between actions. Parallel MDPs [15] consider the combination of local policies based on $Q$ values, however this combination may result in undesirable behavior in some states, which correspond to conflicts between the local policies. The previous approaches in general assume that the subtasks are independent and they do not consider potential conflicts between the policies of each subtask. In this paper we address these problems. First, we present our general architecture and then, the algorithms for solving resource and behavior conflicts.

## III. General architecture

Our work is motivated by robotics, specially by service robots, where frequently it is necessary to simultaneously perform multiple tasks to accomplish certain goal. For instance, if the robot has to deliver a message, it must find and recognize the intended recipient of the message, while simultaneously avoiding obstacles (such as walls or people), and possibly interacting with other persons while navigating in the environment. Although there is much work in developing different abilities or functions for this kind of robots, little attention has been paid for the integration of these behaviors into a complete functional system.

We propose a framework for solving this type of problems based on MDPs. It considers four main phases:

1) Decompose the problem into several subtasks based on functionally, this partition is done by the user. Each subtask is represented by a factored MDP.
2) Solve each MDP independently considering the global objective of the subtask.
3) Combine the local policies and solve resource conflicts. Define restrictions for the behavior conflicts.
4) Execute the local policies in parallel, solving behavior conflicts on–line based on the set of restrictions.

In Fig. 1 we show the main phases of the proposed solution. Next we describe each phase in more detail.

## IV. Functional decomposition

Our focus is on functional design –the process of breaking a system into interacting subcomponents based on functionality. In robotics this decomposition is natural, for example: navigation, vision, interaction and manipulation. Each function contributes to a common objective. For example, a tour guide robot executes several movements to advance (navigation function), gives instructions to a person (interaction function), identifies other persons or places of interest (vision function), if necessary, it open doors or takes a leaflet to give to visitors (manipulation function). All these functions are running at the same time focusing on a particular objective, and at the same time they all contribute to the global goal.

In the first phase the global problem is divided in $k$ functions. Each function has its own objective, and it is modeled as an MDP, named $subMDP$. A $subMDP$ has its own state space, action set, transition and reward functions: $subMDP_i = \{S_i, A_i, \Phi_i, R_i\}$.

The state space of the global problem $S$ is modeled by a set of $n$ variables $X = x_1, .., x_n$ . A factored representation is used for each $subMDP$, so the $subMDP$ state space is modeled by a set of $v$ state variables: $X_i = \{x_1, .., x_v\}$, where $v < n$. There could be common state variables between 2 or more $subMDPs$. The space state of the global problem is fully modeled when the $k$ $subMDPs$ consider the $n$ variables, i.e. $X = \bigcup_{i=1}^{k} X_i$. If the global problem is not modeled in factored representation, the union of all space states of the $subMDPs$ must be equal to the state space of the global problem, i.e., $S = \{S_1 \cup S_2 \cup ... \cup S_k\}$.

The $subMDPs$ are solved independently to obtain the value function ($V_i^*$) and the optimal policy ($\pi_i^*$) for each $subMDP$. In our current implementation to solve each $subMDPs$ we use SPUDD [8]. Once the local policies for all the $subMDP$

are obtained, their solutions are combined to obtain a global policy.

So the global problem is now defined by $k$ $subMDPs$: $subMDP_1, subMDP_2, ..., subMDP_k$, and can be specified as follow:

- State space: $S = \{S_1 \cup S_2 \cup ... \cup S_k\}$. The global state space $S$ is modeled in factored form, where the total state variables is the union of the $k$ sets of state variables $X = \{X_1 \cup X_2 \cup ... \cup X_k\}$.
- Action space: $A = \{A_1, A_2, ..., A_k\}$
- Transition functions: $\Phi = \{\Phi_1, \Phi_2, ..., \Phi_k\}$.
- Reward functions: $R = \{R_1 + R_2 + ... + R_k\}$

An initial global solution is obtained by combining the local optimal policies for each $subMDP$. We ensure a global solution because in each state $s$ all the $subMDPs$ have an action for this state, this is because each $subMDP$ considers one state variable at least, and each state of the global problem is modeled by all state variables.

However, when we combine the solutions of the $subMDPs$ to obtain a global policy, conflicts might arise. In the following section we describe how we detect and solve these conflicts.

## V. Solving policy conflicts

Once the $subMDP$ for each subtask is solved, in principle their policies can be executed simultaneously to solve the global problem. That is, at each time state $(s_i)$, each $subMDP$ selects an action to execute according to its local optimal policy: $\pi^*_1(s_i) = a_1$, $\pi^*_2(s_i) = a_2$, ..., $\pi^*_k(s_i) = a_k$. This set of actions are denoted by $A_{s_i} = \{a_1, a_2, ..., a_k\}$. We have identified two types of conflicts when two or more actions are executed at the same time:

1) *Resource conflicts*. In this case more than one $subMDP$ require the same physical resource (motors, camera, etc.) to execute a different action;
2) *Behavior conflicts*. It occurs when two or more actions require different resources but if they are executed at the same time it results in an undesirable behavior.

### A. Resource conflicts

This type of conflict arises when two or more actions requires the same resource, so it is impossible to execute them at the same time. For example, suppose that a robot is on track to deliver an urgent message, but the battery is running low. In this state, the $navigation$ function may want to turn left to reach the target point, while the $energy$ function may want to turn right to go to recharge the battery. A resource conflict between two actions occurs when $\pi^*_j(s_i) \neq \pi^*_l(s_i)$, and both require the same resource to execute them.

This type of conflicts are solved off–line via a two-phase process. The algorithm for solving resource conflicts is depicted in Algorithm 1. First, it initializes two arrays, $\pi_{initial}$ and $states_{inconflict}$, to save the global policy and the states in conflict, respectively.

An initial global policy is obtained by combining the local policies (lines 4 to 8), such that if there is a conflict between the actions selected by each $subMDP$ for certain state, the one with maximum value is considered (line 8), and the state is marked as a *conflict* state (lines 9 and 10 ). Thus, the initial policy $(\pi_{initial})$ is obtained by: (i) obtaining a global state vector, $S$, which is the union of the local state variables of each $subMDP$; (ii) for each state, $s_i$: if the optimal action is the same for all $subMDPs$, assign this action; otherwise, assign the action with greatest expected value and mark the states with conflicts in their actions.

This initial solution is improved in a second phase (see Algorithm 2) using policy iteration [14]. The previous policy is the initial policy (lines 1 and 2), and considers only the states marked as conflicts (line 3) to improve the initial policy (lines 4 to 12). With these considerations the time complexity is drastically reduced.

The main differences with policy iteration are: (i) it starts from the initial policy $(\pi_{initial})$ instead of a random policy, (ii) it only visits the states marked as conflict. Algorithm 2 returns the final policy (line 13) that solves resource conflicts; in this case no simultaneous actions are allowed.

**Algorithm 1** RCSolver($sMDP_1, sMDP_2, ..., sMDP_k$)

```
1:  $MDP = \{subMDP_1 \cup subMDP_2 \cup ..., \cup subMDP_k\}$
2:  do $\pi_{initial}(s) = a$ for all states $s \in S$
3:  $states_{inconflict} = NULL$
4:  for each state $s_i \in S$ do
5:      if all $\pi^*_k(s_i) = a_j$ then
6:          $\pi_{initial}(s_i) = a_j$
7:      else
8:          $\pi_{initial}(s_i) =$ action $a$ with the largest accumulated $V^a_k(s_i)$.
9:          if $V^\cdot_k(s_i) \neq 0$ then
10:             add the $s_i$ state in $states_{inconflict}$
11:         end if
12:     end if
13: end for
14: $\pi^* =$ PItoRC$(S, A, \Phi, R, \gamma, \pi_{initial})$.
```

**Algorithm 2** PItoRC $(S, A, \Phi, R, \gamma, \pi_{initial}, states_{inconflict})$

```
1:  Let $\Pi = \pi_{initial}$ be the initial policy
2:  Compute $V_\Pi(S)$
3:  for each $s \in states_{inconflict}$ and each $a \in A$ do
4:      if $V_a(s) > V_\Pi(s)$ then
5:          $\Pi^{'}(s) = a$
6:      else
7:          $\Pi^{'}(s) = \Pi(s)$
8:      end if
9:  end for
10: if $\Pi^{'}(s) \neq \Pi(s)$ for any $s$ then
11:     go to 2
12: end if
13: Return $\Pi$
```

## B. *Behavior conflicts*

The user defines off–line a set of restrictions, actions that can not be executed concurrently. Behavior conflicts are solved on–line, the agent check the set of restrictions specified by the user in each step. If there are no restrictions, all the actions are executed concurrently; otherwise, an arbiter selects the action(s) with higher expected utility. The same conflict can occur in several states, however the action selected is not always the same. When there are not conflicts, all $subMDPs$ are fully independent, the global solution is optimal because the local policies are optimal and we consider an additive reward function $R = \{R_1 + R_2 + ... + R_k\}$. To resolve behavior conflicts we also consider two steps:

1) Define restrictions (off–line).
2) Execute concurrent actions (on–line).

*1) Restrictions:* In this phase, the user establishes a set of restrictions in terms of pairs of actions (from different $subMDPs$) that should not be executed at the same time. We consider that given a functional decomposition of the problem and domain knowledge, it is relatively easy for a person to specify these restrictions. For example, in the robotics domain, imagine that the robot is interacting with a user, while the *navigation* module is trying to evade an obstacle (in this case the same person). Therefore these two actions should not run at the same time, and the system must choose which one to run first. The restrictions set (**RS**) lists the actions pairs that can not be executed at the same time.

*2) Concurrent Actions:* Once we have define **RS**, and found the optimal policies for all $subMDPs$, we execute the $k$ $subMDPs$ simultaneously. At each time step, $t$, the agent consults each optimal policy ($\pi_i^*$) of the $k$ $subMDPs$. If there are no restrictions between the actions for all $subMDPs$, all actions are performed concurrently. Otherwise, if there are pairs of actions in conflict, the system selects the action from each conflict pair that maximizes the expected reward for the $subMDP$. The action selected depends on the state in which the conflicts occurs. The selected action is executed concurrently with the rest of actions that have no conflict. The $on-line$ phase is illustrated in Fig. 2.

This strategy assumes that the rewards for each $subMDP$ are given in similar scales, otherwise the comparison will not be fair.

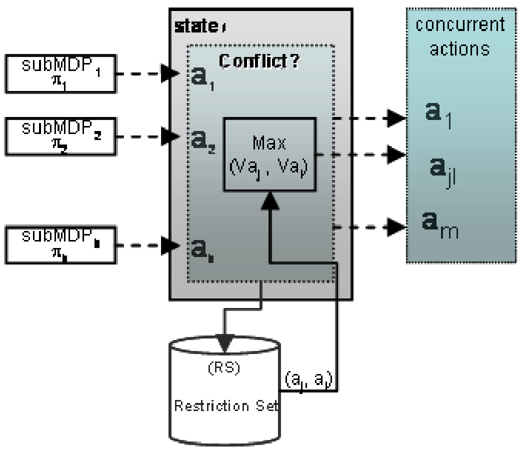

Fig. 2. In an on–line phase the $k$ $subMDPs$ are executed simultaneously. The agent consults each one at each step, finds out if there are conflicts and determines which actions are executed.

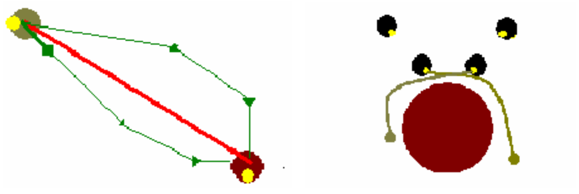

Fig. 3. *Go to* (left) and *obstacle avoidance* (right) functions.

## VI. Experiments

This work is motivated by robotics, in particular service robots. Our initial experiments are in simulation, although we are considering a future implementation in a real robot. We present two sets of experiments: (i) a simulated robot navigation problem, with *resource conflicts*, and (ii) a simulated robot in a message delivery task, with *behavior conflicts*.

## A. *Robot navigation: solving resource conflicts*

To test our approach we consider a robot navigation task. In this test we evaluate the mechanism to solve resource conflicts. The ability to navigate is one of the most important for a mobile robot. The navigation task consists of going from a start position to a target position, safely. For this task it is considered that the robot has a 2D map of the environment and laser and sonar sensors.

We break the navigation task into two interacting subcomponents based on functionality. On subtask (Goto) the robot navigates from its current position to the target position in a straight line, assuming that there are no obstacles (see Fig. 3 left). The second subtask avoids obstacles in the trajectory of the robot, without explicit knowledge of the goal; the obstacles are of different sizes and forms (see Fig. 3 right).

*Goto function:* The robot has to reach a target point from its current position. The idea is that the robot should be directed towards the goal at each step, then the robot advances and if the robot looses its direction toward the goal, its orientation should be corrected, until it reaches the target. For this task we define the following $subMDP$:

- $S$ = $\{(s_{g1})$ oriented, $(s_{g2})$ disoriented, $(s_{g3})$ approaching the goal, $(s_{g4})$ departing for the goal, $(s_{5g})$ close to the goal, $(s_{g6})$ in the target point$\}$
- $A$ = $\{$(A) advance, (S) stop, (D) turn towards the goal$\}$
- $\Phi$ = the uncertainty in the actions is given by assuming that the robot has an 80% chance of doing the desired action correctly, and 20% of doing on alternative action[1].
- $R$ = the largest positive reward is received when the robot reaches the goal. If the action executed moves the robot

[1]In real robots there is always uncertainty in the motion of the robot with respect to the motion commands; the error depends on the type of robot and the surface when it is navigating.

closer to the goal, it receives a small positive reward: and if the action moves it away from the goal, the robot receives a negative reward.

To solve the MDP we use policy iteration, and obtain the optimal policy shown in Table I.

TABLE I
OPTIMAL LOCAL POLICY, $\pi_g^*$, FOR THE GOTO $subMDP$

| state | $s_{g1}$ | $s_{g2}$ | $s_{g3}$ | $s_{g4}$ | $s_{g5}$ | $s_{g6}$ |
|---|---|---|---|---|---|---|
| $\pi_{goto}^*$ | A | D | A | D | S | S |
| $V_{goto}^*$ | 4.788 | 5.338 | 7.252 | 6.259 | 8.781 | 10 |

*Obstacle avoidance function:* The obstacle avoidance function avoids obstacles by going around them without collision. For this task we define the next $subMDP$:

- $S$ = {($s_{a1}$) oriented with obstacle, ($s_{a2}$) oriented without obstacle, ($s_{a3}$) disoriented with obstacle, ($s_{a4}$) disoriented without obstacle, ($s_{a5}$) collided}
- $A$ = {(A) advance, (S) stop, (T) turn,(D) directed towards}
- $\Phi$: the uncertainty in the actions is given by assuming that the robot has an 80% chance of performing the desired action correctly, and 20% in other case.
- $R$: a fixed positive reward is received when the robot avoids the obstacle, if the robot collides it receives the same negative reward.

The optimal local policy for this $subMDP$ is shown in Table II.

TABLE II
OPTIMAL LOCAL POLICY, $\pi_o^*$ FOR THE OBSTACLE AVOIDANCE $subMDP$

| state | $s_{a1}$ | $s_{a2}$ | $s_{a3}$ | $s_{a4}$ | $s_{a5}$ |
|---|---|---|---|---|---|
| $\pi_{avoid}^*$ | A | T | D | T | S |
| $V_{avoid}^*$ | 10 | 6.554 | 8.07 | 7.086 | -10 |

*Combining the $subMDPs$ for Navigation:* To combine both $subMDPs$ we first define the global state vector. There are in total 11 states, six from the goto $subMDP$ and five from obstacle avoidance $subMDP$. However there are some common states. The states $s_{g1}$ corresponds to states $s_{a1}, s_{a2}$, we will call these $s_{ga1}, s_{ga2}$; and $s_{g2}$ to $s_{a3}, s_{a4}$, we will call these $s_{ga3}, s_{ga4}$. So we have only 9 states **S**={$s_{ga1}, s_{ga2}, s_{ga3}, s_{ga4}, s_{g3}, s_{g4}, s_{g5}, s_{g6}, s_{a5}$}. Then we obtain the initial global policy, $\pi_{initial}$. The process to generate it and the resulting $\pi_{initial}$ are shown in Table III. Finally we apply PItoRC and obtain the global optimal policy, this is shown in Table IV.

To validate this result, we solved a single MDP that considers the goto and obstacle avoidance function, and the resulting policy is the same as the one shown in Table IV. We also evaluated experimentally this policy in different simulated environments, changing the goal positions and obstacle distributions. In Fig. 4 we show some results of the *goto* $subMDP$, and in Fig. 5 of *obstacle avoidance* $subMDP$.

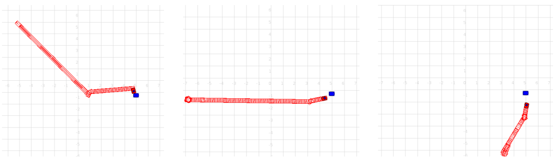
Fig. 4. Some results of the *goto* policy

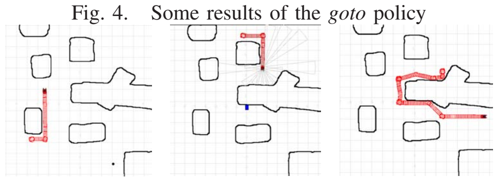
Fig. 5. Some results of the *obstacle avoidance* policy

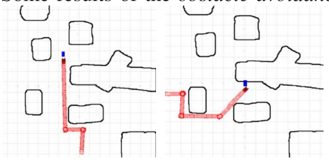
Fig. 6. Some results of the *global navigation* policy

Several results form the global navigation policy are depicted in Fig. 6.

## B. Human–robot interaction: solving behavior conflicts

The next set of experiments considers a simulated messenger robot interacting with persons in its environment. The objective of this experiment is to test the approach for solving behavior conflicts. We use the result obtained above for the navigation function and combine them with a *human–robot interaction* function.

The *human–robot interaction* function is defined and solved as another $subMDP$. We defined 10 states and 6 actions: (W) waits for interaction, (O) offer service, (H) hello, (E) excuse, (G) get message, (Dm) deliver message. The local policy obtained is shown in Table V.

TABLE V
LOCAL POLICY, $\pi_h^*$, FOR THE *human–robot interaction* $subMDP$.

| state | $s_{i1}$ | $s_{i2}$ | $s_{i3}$ | $s_{i4}$ | $s_{i5}$ | $s_{i6}$ | $s_{i7}$ | $s_{i8}$ | $s_{i9}$ | $s_{i10}$ |
|---|---|---|---|---|---|---|---|---|---|---|
| $\pi_{IHR}^*$ | W | W | O | H | G | Dm | Sa | E | H | O |

Next we describe a set of restrictions for the navigation vs. human–robot interaction functions. This restriction set **RS** is defined by the action pairs listed in table VI.

We evaluated experimentally the execution of both local policies, for navigation and interaction, concurrently, with the previously defined set of restrictions. The tests considered different simulated environments, changing the users and goals positions. We run both policies simultaneously, and these were consulted at each state. The robot starts at a random position, and has no message to deliver. We simulated persons in the environment that can be moved by the user. Any person can interact with the robot at any time during the experiment. The restrictions are evaluated at each state, and if a conflict

TABLE III

OBTAINING THE $\pi_{initial}$ FROM TWO $subMDPs$ FOR NAVIGATION. THE FIRST TWO ROWS SHOW THE POLICY AND VALUE FUNCTIONS OF EACH SUBMDP, THE THIRD ROW SHOWS THE INITIAL POLICY AND THE LAST ROW SHOWS THE STATES WITH CONFLICTS.

| | $s_{ga1}$ | $s_{ga2}$ | $s_{ga3}$ | $s_{ga4}$ | $s_{g3}$ | $s_{g4}$ | $s_{g5}$ | $s_{g6}$ | $s_{a5}$ |
|---|---|---|---|---|---|---|---|---|---|
| $\pi_{goto}$ | A | A | D | D | A | D | S | S | A |
| $V_{goto}$ | 4.78 | 4.78 | 5.33 | 5.33 | 7.25 | 6.25 | 8.78 | 10 | 0 |
| $\pi_{avoid}$ | A | T | D | T | A | A | A | A | S |
| $V_{avoid}$ | 10 | 6.55 | 8.0 | 7.08 | 0 | 0 | 0 | 0 | -10 |
| $\pi_{ini}$ | A | T | D | T | A | D | S | S | S |
| $s_{inconf}$ | | X | | X | | | | | X |

TABLE IV

FINAL GLOBAL POLICY FOR THE NAVIGATION TASK.

| | $s_{ga1}$ | $s_{ga2}$ | $s_{ga3}$ | $s_{ga4}$ | $s_{g3}$ | $s_{g4}$ | $s_{g5}$ | $s_{g6}$ | $s_{a5}$ |
|---|---|---|---|---|---|---|---|---|---|
| $\pi^*_{global}$ | A | T | D | T | A | D | S | A | T |
| $V^*_{global}$ | 19 | 15 | 19 | 17 | 16 | 15 | 18 | 20 | 0 |

TABLE VI

RESTRICTION SET **RS** FOR THE MESSENGER ROBOT.

| **navigation** | **human–robot interaction** |
|---|---|
| (A) advance | (O) offer service |
| (T) turn | (O) offer service |
| (D) directed towards | (O) offer service |
| (A) advance | (G) get message |
| (T) turn | (G) get message |
| (D) directed towards | (G) get message |
| (A) advance | (Dm) deliver message |
| (T) turn | (Dm) deliver message |
| (D) directed towards | (Dm) deliver message |
| (A) advance | (E) excuse |

arises, the action with greatest expected value is selected. We observed in the experiments that different actions were chosen depending on the state in which the conflict occurred.

In Fig. 7 we illustrate one of the experiments. The robot begins at the point marked with *start position*, the *navigation* function wants to reach the *goal point*, while *HRI* function wants to catch a petition or interact with a user. The line is the path followed by the robot. The triangles show the same behavior conflict in different states, the actions in conflict set are (A) advance and (G) get message. We can notice that in the first case the robot decides to interact with the user and obtain a petition, while in the second case the robot ignores the user and advances to the goal. In several cases the user only salutes the robot which responds and advances at the same time (shown with a circle).

## VII. CONCLUSIONS AND FUTURE DIRECTIONS

We have presented a framework for solving complex Markov decision processes based on functional decomposition which partitions the problem in several simpler MDPs. We obtain the optimal policy for each $subMDP$, and then combine the results to obtain a global solution such that the actions of each $subMDP$ are executed concurrently. We have identified two types of conflicts: resource and behavior conflicts. For resource conflicts the local policies are combined to obtain an initial policy which is refined using a modified policy interaction algorithm. For behavior conflicts a set of user-defined restrictions are used to identified conflicts and the action with highest expected value is selected. Experiments with a simulated robot with both types of conflicts show the feasibility of the proposed approach.

We are currently testing our approach in a more complex experiment. The global problem has a larger state space, including more functions thus increasing the conflicts between local policies. The objective of the messenger robot, is to pickup and deliver more than one object or message between users in an office environment. Users must be detected and identified by vision. The system has to plan a delivery schedule, navigate safely and interact with the users during its path. While the robot is delivering, it can also guide a visitor to a particular office. During the task an animated face expresses the robot emotions. For these new experiments we are combining four $subMDPs$: (i) navigation, (ii) interaction, (iii) vision, and (iv) facial expression.

This will be implemented and tested on a real mobile robot. We are also working on a more formal analysis of the optimality of the proposed approach, and in extending the syntaxis uses to specify the restrictions.

## REFERENCES

[1] R. Iris Bahar, Erica A. Frohm, Charles M. Gaona, Gary D. Hachtel, Enrico Macii, Abelardo Pardo, and Fabio Somenzi. Algebraic decision diagrams and their applications. In *ICCAD*, pages 188–191, 1993.

[2] Richard Bellman. Dynamic programming and stochastic control processes. *Information and Control*, 1(3):228–239, 1958.

[3] Craig Boutilier, Richard Dearden, and Moisés Goldszmidt. Exploiting structure in policy construction. In *IJCAI*, pages 1104–1113, 1995.

[4] Thomas Dean and Robert Givan. Model minimization in markov decision processes. In *AAAI/IAAI*, pages 106–111, 1997.

[5] Thomas G. Dietterich. An overview of maxq hierarchical reinforcement learning. In *SARA*, pages 26–44, 2000.

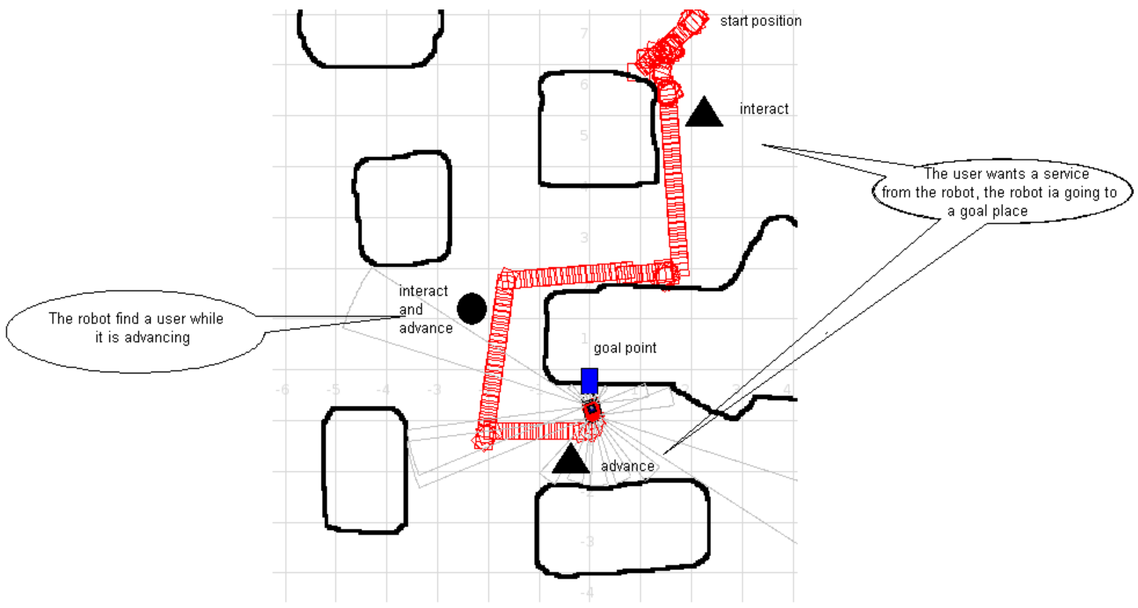
start position
interact
The user wants a service from the robot, the robot ia going to a goal place
interact and advance
The robot find a user while it is advancing
goal point
advance

Fig. 7. The messenger robot task. The robot navigates from start position to goal point, while it is navigating finds a user who wants a service from the robot, the system detects a behavior conflicts in two different states (triangles marked) between navigation and interact actions, the system selects an action based on RC. The circle show the execution of two concurrent actions, the robot while delivering the message finds a user and salutes it.

[6] P. Elinas, L.E. Sucar, A. Reyes, and J. Hoey. Decision theoretic approach for task coordination in social robots. In *IEEE International Workshop on Robot and Humand Interaction Communication*, pages 679–684, 2004.

[7] Zoubin Ghahramani. Learning dynamic bayesian networks. In *Adaptive Processing of Sequences and Data Structures*, pages 168–197. Springer-Verlag, 1998.

[8] Jesse Hoey, Robert St-Aubin, Alan J. Hu, and Craig Boutilier. Spudd: Stochastic planning using decision diagrams. In *UAI*, pages 279–288, 1999.

[9] Nicholas K. Jong and Peter Stone. State abstraction discovery from irrelevant state variables. In *IJCAI*, pages 752–757, 2005.

[10] Pierre Laroche, Yann Boniface, and René Schott. A new decomposition technique for solving markov decision processes. In *SAC*, pages 12–16, 2001.

[11] Lihong Li, Thomas J. Walsh, and Michael L. Littman. Towards a unified theory of state abstractions for mdps. In *In Proceedings of the Ninth International Symposium on Artificial Intelligence and Mathematics*, pages 531–539, 2006.

[12] Nicolas Meuleau, Milos Hauskrecht, Kee-Eung Kim, Leonid Peshkin, Leslie Pack Kaelbling, Thomas Dean, and Craig Boutilier. Solving very large weakly coupled markov decision processes. In *AAAI/IAAI*, pages 165–172, 1998.

[13] Ronald Parr and Stuart Russell. Reinforcement learning with hierarchies of machines. In Michael I. Jordan, Michael J. Kearns, and Sara A. Solla, editors, *Advances in Neural Information Processing Systems*, volume 10. The MIT Press, 1997.

[14] Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley-Interscience, April 1994.

[15] L.E. Sucar. Parallel markov Decision Processes. In P. Lucas, J. Gamez, and A. Salmeron, editors, *Advances in Probabilistic graphycal Models*, pages 295–309. Springer, 2007.

[16] Ahmed Y. Tawfik and Eric Neufeld. Temporal bayesian networks. In *TIME*, pages 85–92, 1994.