An Integrated Approach of Learning, Planning, and Execution

RAMÓN GARCÍA-MARTÍNEZ
Departamento de Computación, Facultad de Ingeniería, Universidad de Buenos Aires, Bynon 1605, Adrogue (1846), Buenos Aires, Argentina; e-mail: rgm@mara.f .uba.ar

DANIEL BORRAJO
Departamento de Informática, Universidad Carlos III de Madrid, Avda. de la Universidad, 30, 28911 Leganés, Madrid, Spain; e-mail: dborrajo@ia.uc3m.es

Abstract. Agents (hardware or software) that act autonomously in an environment have to be able to integrate three basic behaviors: planning, execution, and learning. This integration is mandatory when the agent has no knowledge about how its actions can affect the environment, how the environment reacts to its actions, or, when the agent does not receive as an explicit input, the goals it must achieve. Without an "a priori" theory, autonomous agents should be able to self-propose goals, set-up plans for achieving the goals according to previously learned models of the agent and the environment, and learn those models from past experiences of successful and failed executions of plans. Planning involves selecting a goal to reach and computing a set of actions that will allow the autonomous agent to achieve the goal. Execution deals with the interaction with the environment by application of planned actions, observation of resulting perceptions, and control of successful achievement of the goals. Learning is needed to predict the reactions of the environment to the agent actions, thus guiding the agent to achieve its goals more effi iently.

In this context, most of the learning systems applied to problem solving have been used to learn control knowledge for guiding the search for a plan, but few systems have focused on the acquisition of planning operator descriptions. As an example, currently, one of the most used techniques for the integration of (a way of) planning, execution, and learning is reinforcement learning. However, they usually do not consider the representation of action descriptions, so they cannot reason in terms of goals and ways of achieving those goals.

In this paper, we present an integrated architecture, LOPE, that learns operator def nitions, plans using those operators, and executes the plans for modifying the acquired operators. The resulting system is domain-independent, and we have performed experiments in a robotic framework. The results clearly show that the integrated planning, learning, and executing system outperforms the basic planner in that domain.

Key words: autonomous intelligent systems, embedded machine learning, planning and execution, reinforcement learning, theory formation, theory revision, unsupervised machine learning.

1. Introduction

Autonomous intelligent behavior is an area with an emerging interest within Artif cial Intelligence researchers (Fritz et al., 1989; Mahavedan and Connell, 1992;

Simmons and Mitchell, 1989; Stone and Veloso, 1998; Matellán et al., 1998; Ashish et al., 1997), ranging from work on autonomous robotic agents to Web-based software agents. It integrates many areas, such as robotics, planning, and machine learning. This integration opens many questions that arise when designing such systems, such as how operator descriptions can be *incrementally* and *automatically* acquired from the planning/execution cycle, or how a planner can use incomplete and/or incorrect knowledge, as mentioned in (Wang, 1996). With respect to learning, autonomous systems must generate theories of how their environment reacts to their actions, and how the actions affect the environment. Usually, these theories are partial, incomplete and incorrect, but they can be used to plan, to further modify those theories, or to create new ones.

Among the different types of machine learning techniques, those based on observation and discovery are the best modelers for human behavior (Falkenhainer, 1990). Thus, it is interesting to study how an autonomous system can automatically build planning operators that model its environment (Fritz et al., 1989; García-Martínez, 1993; García-Martínez, 1997). In this context, machine learning applied to planning has mainly focused on learning control knowledge in many different ways such as: macrooperators (Fikes et al., 1972), control rules (Minton, 1988; Borrajo and Veloso, 1997), control knowledge into operator descriptions (Langley, 1983) or cases (Veloso, 1994). There is also currently a big trend on learning which actions to apply in any state in the context of reinforcement learning (Sutton, 1990; Watkins and Dayan, 1992). However, very few have approached the generalized operators acquisition problem (Carbonell and Gil, 1990; Wang, 1996), which is crucial when dealing with systems that must *autonomously* adapt to a changing environment.

We present in this paper a system, LOPE,[⋆] that integrates planning, learning, and execution in a closed loop, showing an autonomous intelligent behavior. Learning planning operators is achieved by observing the consequences of executing planned actions in the environment (García-Martínez, 1993).[⋆⋆] In order to speed up the convergence, heuristic generalizations of the observations have been used. Also, probability distribution estimators have been introduced to handle the contradictions among the generated planning operators. The learning mechanism allows not only to acquire operator descriptions, but also to adapt those descriptions to changes in the environment. The results show how the learning mechanism outperforms the behavior of the base planner with respect to successful plans (plans that achieve self-proposed goals). We also present an extension of the learning mechanism that allows knowledge to be shared among several LOPE agents. The results of using this multi-agent scheme show how the interaction with other learning agents greatly improves learning convergence and successful behavior.

[⋆] LOPE stands for Learning by Observation in Planning Environments.

[⋆⋆] When we talk about environment, we do not refer to a unique setup, but to the generic concept of environment.

Section 2 describes the general architecture of the LOPE system, def ning its architecture and top-level algorithm. Section 3 def nes the representation that will be used in the paper for situations, observations and planning operators. Section 4 presents the learning model and its components (heuristic generalizations and probabilities estimation). Section 5 def nes the planner and how probabilities estimations allow to perform a stochastic analysis for each plan. Section 6 describes how several LOPE agents can cooperate to converge faster to an environment model. Section 7 explains the performed experiments and their results that show how the overall behavior outperforms a simpler planner. Section 8 describes the relation with other approaches. And, f nally, Section 9 draws the conclusions of the work.

2. General System Description

The objective of the LOPE system is to autonomously plan for achieving self-proposed goals, executing plans, f nding out deviations from the plans or correct behavior, and learning operators (models) that predict the effects of actions in the environment, by observing the consequences of those actions. The system can be described as an agent that receives perceptions from the environment, called *situations*, applies actions, and learns from its interaction with the world. Figure 1 shows a schematic view of the architecture, and Figure 2 presents the high level description of the algorithm. Sections 4 and 5 will present in more detail the functions `learning` and `plan`, respectively.

At the beginning, the system does not have any knowledge, so it perceives the initial situation, and selects a random action to execute in the environment. Then, it loops by executing an action, perceiving the resulting situation and utility of the situation (explained in Section 3), learning from observing the effect of applying

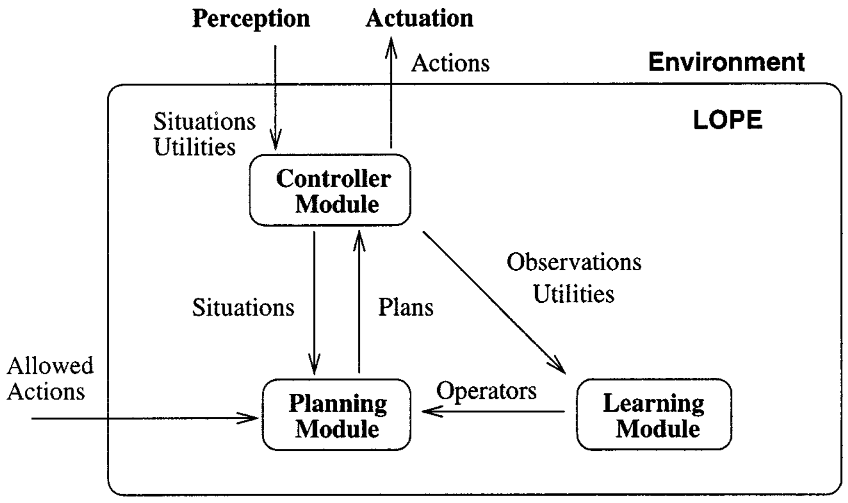

Figure 1. Architecture of the integrated system.

```
Function Lope ($n$, $\mathcal{A}$)

$n$: number of execution cycles
$\mathcal{A}$: Set of possible actions
$S$, $S'$: Perceived situations
$\mathcal{O}$: Set of learned operators, initially empty
$P$: Current plan in execution, initially empty

$S$:=perceive-situation
$A$:=random-selection($\mathcal{A}$)
Repeat $n$ times
   execute-action($A$)
   $S'$:=perceive-situation
   $U$:=perceive-utility
   $\mathcal{O}$:=learning($S$, $A$, $S'$, $U$, $\mathcal{O}$,false)
   If $P = \emptyset$ OR operator.conditions($O$)$\not\subseteq S'$
   Then $P$:=plan($S'$, $\mathcal{O}$, $\mathcal{A}$)
   $O$:=pop($P$)
   $A$:=operator.action($O$)
   $S := S'$
```

Figure 2. Top level algorithm of the whole architecture.

the action in the environment, and planning for further interactions with the environment when the previous plan has f nished its execution, or the system observes a mismatch between the predicted situation by the agent operators and the situation it perceived from the environment. In the latter case, it re-plans for achieving a high utility goal from the current situation.

The top-level goal of the planning algorithm is implicit in the system: achieving a situation with the highest utility, so the goal is not an input to the system. This fact does not remove generality to the overall architecture, since the function that computes the utility can be changed (even dynamically) to the one that ref ects other types of goals, as in classical planning.

3. Representation

As it was mentioned before, the architecture is domain-independent. However, in order to explain its behavior, we will refer throughout this paper to the application we developed in a robotic domain. In this domain, the agent plans, learns, and executes in a simulated world according to the robot model described and used by many authors, such as (Mahavedan and Connell, 1992; Shen, 1993). Thus, a model of the environment refers to a mapping between perceived situations, performed actions, and expected new situations. This representation would be closer to the one used in reactive planners (Brooks, 1986), than to high-level models of the environment used by deliberative planners, such as the STRIPS representation (Fikes and Nilsson, 1971). However, we believe that for real intelligent behavior, some kind of collaboration between these two levels is needed, by allowing reactive behavior on one side, and high-level planning on the other. Therefore, we propose an intermediate representation by allowing the basic reactive representation (perceived situation, action, and resulting situation) to be extended by generalized operators, and search-based planning, as discussed later. The autonomous agent type of world that we used for the experiments is a two-dimensional grid, where each position of the grid can have different elements, such as robots (agents), obstacles, energy points, or be empty. For generality purposes, we used many different grids as explained in Section 7.

For LOPE, as for many other systems, there is a difference between the world states, common to classical planning, and the observations it perceives. While classical planners are mainly interested in the high-level descriptions of the states (e.g., on (A,B) in the blocksworld), LOPE builds its operators based on the perceptions it receives from its sensors;⋆ its "states" are the inputs it receives from its sensoring system. Any post-processing of its inputs in order to translate them into high-level descriptions can be done without affecting the overall behavior. Because of the natural limitations of the sensory system, the agents map different states of the environment into a single one, even if they would not be equivalent in terms of the best action to perform. Then, we developed a way of taking this into account, by associating probability estimations to operators. Therefore, our system implicitly manages noisy domains and hidden states.

We are using propositional logic as the underlying representation formalism given that we have found it powerful enough for the type of problem solving that each agent is performing (stochastic planning). Most of the learning algorithms can be easily extended to predicate logic, with the known exponential increase on computational cost. One advantage of propositional logic on the types of tasks we are applying it to (robotic tasks) is that it allows to have reasonable good performance on most tasks in comparison with higher level systems (such as STRIPS).

In previous work of the authors (García-Martínez, 1997), the representation of operators was based on the model proposed in (Fritz et al., 1989), in which an observation (also called experience unit) had the following structure:

[Initial Situation, Action, Final Situation],

⋆ Here, the word sensors refer to the generic idea of input, so this is applicable to non robotic domains.

Table I. Representation of a planning operator

Planning operator: O_i		
Feature	Description	Type of values
C	Initial situation (conditions)	*p-list*
A	Action	*action*
F	Final situation	*p-list*
P	Times that the operator O_i was successfully applied (the expected f nal situation, F, was obtained)	*integer*
K	Times that the action A was applied to C	*integer*
U	Utility level reached applying the action to the initial situation C of the operator	*real* [0, 1]

where initial and f nal situations are lists of propositions that can be either true or false. Observations were directly used as planning operators. In this paper, while the concept of observation does not change, the representation of operators is extended, by the addition of features that allow to determine their planning/execution acceptability. This provides also a solution to noise in sensors/actuators, and the hidden state problem. The proposed planning operator model has the structure and meaning described in Table I, where *p-list, action* and U are domain-dependent and have the following meaning:

– *p-list* is a list of propositions, that can be preceded by the ¬ symbol, denoting negation. If a proposition does not appear on that list, it is assumed that its value does not matter.
– *action* can be any of the set of allowed actions that each agent can perform. For instance, in our robotic domain, it can be one of `go`, `turn-left`, `turn-right`, and `stop`, while in the Robosoccer domain, it would be one of `dash`, `turn`, `kick` or `catch` (Kitano et al., 1995).
– U is a function that measures how useful the current situation is for the agent, and refers implicitly to the distance to the agent's goal (similar concept to the reward in reinforcement learning (Watkins and Dayan, 1992)). As an example, in the performed experiments, it has been measured as a function of the distance of a robot (agent) to the closest energy point, E.

  As shown in Figure 1, this measure is given to the system as an input. This function could be changed to allow different behaviors (achievement of different goals) for the agent. For instance, in a robotic soccer domain, we could have different players with different behaviors, each one depending on their specif c goals (Matellán et al., 1998). Examples of such goals are "being close to the ball", "scoring a goal", or "being in front of an opponent".

The parameters P and K allow the architecture to decrease the effect of noise in the sensors or in the environment, and hidden state problems. This is possible due to the underlying bias of the system which assumes that the environment can be dynamic, noisy, and there can be features of the state that are not captured by the sensoring system, and, therefore, not handled by the planning or learning mechanisms.

4. Learning planning operators

We will f rst def ne the concepts of *similar* and *equal* operators needed for the learning method, to further detail the learning method, present an example, and discuss the generalization heuristics.

4.1. DEFINITIONS

Given two operators $O_1 = [C_1, A_1, F_1, P_1, K_1, U_1]$ and $O_2 = [C_2, A_2, F_2, P_2, K_2, U_2]$, and an observation $o = [S_i, A, S_f]$, we provide the following def nitions:

DEFINITION 1. If the conditions and actions of two operators are equal, then the two operators are `similar`.

$$(C_1 = C_2) \wedge (A_1 = A_2) \rightarrow \texttt{similar-op}(O_1, O_2).$$

DEFINITION 2. If the conditions, actions, and f nal situations of two operators are equal, then the two operators are `equal`.

$$(C_1 = C_2) \wedge (A_1 = A_2) \wedge (F_1 = F_2) \rightarrow \texttt{equal-op}(O_1, O_2).$$

DEFINITION 3. If the actions of an operator and an observation are equal, and the formula describing the initial situation of the observation includes the conditions of the operator (the conditions of the operator are true in the initial situation of the observation)), then the observation is `similar` to the operator.

$$(C_1 \subseteq S_i) \wedge (A_1 = A) \rightarrow \texttt{similar-ob}(o, O_1).$$

For instance, if situations are represented by the presence or not of obstacles in the right (`r`), left (`l`), or in front (`f`) of a robot, and the system observes $o =$ [(`l,f`),`turn`,(`l,r`)], this observation would be `similar` to operator:

$$O_1 = [(\texttt{f}), \texttt{turn}, (\texttt{l}), P_1, K_1, U_1]$$

given that (`f`)$\subseteq$(`l,f`), and both have the same action.

DEFINITION 4. If the actions of an operator and an observation are equal, the initial situation of the observation includes the conditions of the operator, and the f nal situation of the observation includes the f nal situation of the operator (the expected f nal situation of the operator is true in the observed f nal situation), then the observation is `equal` to the operator (conf rms it).

$$(C_1 \subseteq S_i) \wedge (A_1 = A) \wedge (F_1 \subseteq S_f) \rightarrow \texttt{equal-ob}(o, O_1).$$

It can also be def ned in terms of similarity as: an observation is `equal` to an operator if it is `similar` and the expected f nal situation of the operator is true in the observed f nal situation.

$$\texttt{similar-ob}(o, O_1) \wedge (F_1 \subseteq S_f) \rightarrow \texttt{equal-ob}(o, O_1).$$

Following the previous example, the observation would also be equal to the operator (the observation conf rms the operator), given that they are `similar`, and (`l`)⊆(`l,r`).

4.2. HIGH LEVEL LEARNING ALGORITHM

Suppose a situation S_i is perceived by the system, and there is a set of operators, $\mathcal{O}$, such that each operator is of the form $O_i = [C, A, F, P, K, U]$. If the system applies the action A, arriving at a situation S_f, the learning method processes this new observation by the algorithm shown in Figure 3. When a new observation arrives at the learning module, it checks if a similar operator exists.

- If there is such *similar* operator, it checks to see if the observation is *equal* to any operator. Then, it rewards all such operators and punishes *similar* ones. If a *similar* operator exists, but there is none that is *equal* to the observation, it creates a new operator, punishes *similar* operators to the new one, and generalizes those *similar* operators. The operators generated by the generalization procedure reward *equal* operators and punish *similar* ones (recursive call).
- If it does not f nd a *similar* operator for the input observation, it creates a new one.

The function `heuristic-generalization` is explained in more detail in Section 4.4. Function `punish-operator` increments the number of times that the pair (condition,action) of similar operators to the observation (generalized or not) has been observed.* The effect of incrementing their K is equivalent to *punishing* them, since their probability of success decreases. Its algorithm is shown in Figure 5. Similarly, function `punish-similar-operators` punishes all operators in a set that are similar to a new one, including itself (see Figure 4).

Function `reward-operator` increments the P of a successful operator, with the equivalent effect of *rewarding* it, since its probability of success increases. Figure 6 shows its high level description. The algorithm for computing the K and P of

* In single agent conf gurations, it gets incremented in one. In multi-agent conf gurations, it will be greater than one, as explained in Section 6.

```
Function Learning ($o$, $U$, $\mathcal{O}$,generalized?): $\mathcal{O}$

$o$: observation made. $o = [S_i, A, S_f]$
    $S_i$: Initial situation of the observation
    $A$: Applied action of the observation
    $S_f$: Observed f nal situation
$U$: Observed utility
$\mathcal{O}$: Set of operator descriptions
generalized?: true if $o$ is a generalized observation. Otherwise, false

If exists $O_i \in \mathcal{O}$ such that similar-ob($o$, $O_i$)
Then If exists $O_i \in \mathcal{O}$ such that equal-ob($o$, $O_i$)
       Then Forall $O_i \in \mathcal{O}$ such that equal-ob($o$, $O_i$) do
                reward-operator($O_i$,1)
                $U_{O_i}$:=max($U_{O_i}$, $U$)
                Forall $O_j \in \mathcal{O}$ such that similar-ob($o$, $O_j$) do
                   punish-operator($O_j$,1)
       Else $O_n$:=[$S_i$, $A$, $S_f$, 1, $K_{O_i}$, $U$]
            $\mathcal{O}$:=$\mathcal{O} \cup \{O_n\}$
            punish-similar-operators($O_n$, $\mathcal{O}$,1)
            If not(generalized?)
            Then $\mathcal{M}$:=heuristic-generalization($S_i$, $A$, $S_f$, $\mathcal{O}$)
                 Forall $O_m \in \mathcal{M}$ do learning($O_m$, $U$, $\mathcal{O}$,true)
Else $\mathcal{O}$:=$\mathcal{O} \cup \{[S_i, A, S_f, 1, 1, U]\}$
Return $\mathcal{O}$
```

Figure 3. Algorithm that modif es the operators descriptions after having seen a new observation.

each operator preserves some formal properties. For instance, it preserves that the sum of the quotient P/K of all `similar` operators is equal to one, for each pair (condition,action).

With respect to the utility, the function `learning` records, for each operator, the utility of the highest-utility situation achieved by applying the operator action to the operator condition situation.[⋆]

[⋆] Since the fi al situation can be generalized, there might be more than one utility. Only the highest one is stored.

```
Function Punish-similar-operators ($O$, $\mathcal{O}$,$i$)
$O$: New operator
$\mathcal{O}$: Set of operators
$i$: Increment to be applied to operators $K$s
    Forall $O_i \in \mathcal{O}$ such that similar-op($O$, $O_i$) do
      punish-operator($O_i$, $i$)
```

Figure 4. Algorithm that punishes the operators similar to O.

```
Function Punish-operator ($O$,$i$)
$O$: New operator
$i$: Increment to be applied to operators $K$s
    operator.K($O$):=operator.K($O$)+$i$
```

Figure 5. Algorithm that punishes the operator O.

```
Function Reward-operator ($O$, $i$)
$O$: Successful operator
$i$: Increment to be applied to operators $P$s and $K$s
operator.P($O$):=operator.P($O$)+$i$
```

Figure 6. Algorithm that rewards a successful operator.

4.3. EXAMPLE OF LEARNING EPISODES

Let us def ne a domain in which a robot explores a two-dimensional grid with obstacles, and energy points as shown with an example in Figure 7.⋆ Initially, it does not have any knowledge of how the environment will react to the actions that the robot can apply: go and turn. Its only goal will be to maximize the utility (to minimize the Manhattan distance to the closer energy point in this example, E) as def ned in Equation (1).

$$U(S, E) = \frac{1}{|1 + d(S, E)|}, \qquad (1)$$

⋆ This is an oversimplifi ation in terms of size of the grids used in the experiments (700×1000).

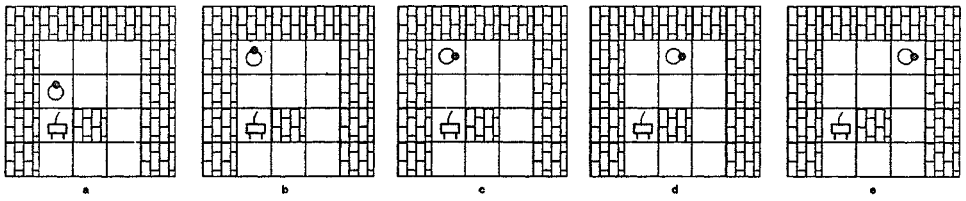

Figure 7. Example of an exploratory robot in a 2-dimensional grid. The robot orientation is represented by the small grey circle. Energy points are represented by plugs.

where S is the robot position, and $d(S, E)$ is the distance between S and E.

Its inputs are what the robot sensors perceive at any moment. As an example, suppose that it can only "see" the positions that are directly to the right (`r`) and left (`l`), and the ones that are directly in front of him: front (`f`), front-left (`fl`), and front-right (`fr`). They will be represented by the propositions that appear in parenthesis. If a proposition appears in any list of propositions, it means that the corresponding sensor has detected an obstacle in that position. If it appears negated, it means that the position is empty. And, if it does not appear in the list, it means that it does not matter what it contains.

We will see now how the system builds a set of operators from the following observations o_1, o_2, o_3 and o_4, whose graphical representation can be seen in Figure 7(a)–(e).

$$o_1 = [(\mathtt{l}, \mathtt{fl}, \neg\mathtt{r}, \neg\mathtt{f}, \neg\mathtt{fr}), \mathtt{go}, (\mathtt{l}, \mathtt{fl}, \neg\mathtt{r}, \mathtt{f}, \mathtt{fr})], U = 1/3,$$
$$o_2 = [(\mathtt{l}, \mathtt{fl}, \neg\mathtt{r}, \mathtt{f}, \mathtt{fr}), \mathtt{turn}, (\mathtt{l}, \mathtt{fl}, \neg\mathtt{r}, \neg\mathtt{f}, \neg\mathtt{fr})], U = 1/3,$$
$$o_3 = [(\mathtt{l}, \mathtt{fl}, \neg\mathtt{r}, \neg\mathtt{f}, \neg\mathtt{fr}), \mathtt{go}, (\mathtt{l}, \mathtt{fl}, \neg\mathtt{r}, \neg\mathtt{f}, \neg\mathtt{fr})], U = 1/4,$$
$$o_4 = [(\mathtt{l}, \mathtt{fl}, \neg\mathtt{r}, \neg\mathtt{f}, \neg\mathtt{fr}), \mathtt{go}, (\mathtt{l}, \mathtt{fl}, \neg\mathtt{r}, \mathtt{f}, \mathtt{fr})], U = 1/5.$$

After observing o_1, according to the algorithm in Figure 3, it generates a new operator:

$$O_{G1} = \left[(\mathtt{l}, \mathtt{fl}, \neg\mathtt{r}, \neg\mathtt{f}, \neg\mathtt{fr}),\ \mathtt{go},\ (\mathtt{l}, \mathtt{fl}, \neg\mathtt{r}, \mathtt{f}, \mathtt{fr}), 1, 1, 1/3\right].$$

The observation o_2 refers to another action, `turn`, which defnes a new operator:

$$O_{T1} = \left[(\mathtt{l}, \mathtt{fl}, \neg\mathtt{r}, \mathtt{f}, \mathtt{fr}),\ \mathtt{turn},\ (\mathtt{l}, \mathtt{fl}, \neg\mathtt{r}, \neg\mathtt{f}, \neg\mathtt{fr}), 1, 1, 1/3\right].$$

When it observes o_3, it fnds out that there is a similar operator, O_{G1}, (equal conditions and action, but differs in the predicted effects). Thus, it frst includes the new operator

$$O_{G2} = \left[(\mathtt{l}, \mathtt{fl}, \neg\mathtt{r}, \neg\mathtt{f}, \neg\mathtt{fr}),\ \mathtt{go},\ (\mathtt{l}, \mathtt{fl}, \neg\mathtt{r}, \neg\mathtt{f}, \neg\mathtt{fr}), 1, 1, 1/4\right]$$

into the set of operators $\mathcal{O}$. Then, it punishes all similar operators, including itself (O_{G1} and O_{G2} in this case), changing them to be:

$$O_{G1} = \left[(\mathtt{l}, \mathtt{fl}, \neg\mathtt{r}, \neg\mathtt{f}, \neg\mathtt{fr}),\ \mathtt{go},\ (\mathtt{l}, \mathtt{fl}, \neg\mathtt{r}, \mathtt{f}, \mathtt{fr}), 1, 2, 1/3\right],$$
$$O_{G2} = \left[(\mathtt{l}, \mathtt{fl}, \neg\mathtt{r}, \neg\mathtt{f}, \neg\mathtt{fr}),\ \mathtt{go},\ (\mathtt{l}, \mathtt{fl}, \neg\mathtt{r}, \neg\mathtt{f}, \neg\mathtt{fr}), 1, 2, 1/4\right].$$

After this, it calls the generalization heuristics to create generalized observations, as described in the following section. The inputs to the generalization procedure are the initial situation, $(\texttt{l}, \texttt{fl}, \neg\texttt{r}, \neg\texttt{f}, \neg\texttt{fr})$, the action, go, the f nal situation, $(\texttt{l}, \texttt{fl}, \neg\texttt{r}, \neg\texttt{f}, \neg\texttt{fr})$, and the set of operators $\mathcal{O} = \{O_{G1}, O_{T1}, O_{G2}\}$. Among other heuristics, the retraction generalization heuristic would generate the following generalized observation $m = [(\texttt{l}, \texttt{fl}, \neg\texttt{r}, \neg\texttt{f}, \neg\texttt{fr}), \texttt{go}, (\texttt{l}, \texttt{fl}, \neg\texttt{r})]$. Then, it recursively calls the `learning` function, and, since it now f nds that there are similar (but not equal) operators, O_{G1} and O_{G2}, it adds a new (generalized) operator,

$$O_{G3} = [(\texttt{l}, \texttt{fl}, \neg\texttt{r}, \neg\texttt{f}, \neg\texttt{fr}),\ \texttt{go},\ (\texttt{l}, \texttt{fl}, \neg\texttt{r}), 1, 2, 1/4\big],$$

and punishes all similar operators by incrementing their K, leaving $\mathcal{O}$ as:

$$\begin{aligned}
O_{G1} &= \big[(\texttt{l}, \texttt{fl}, \neg\texttt{r}, \neg\texttt{f}, \neg\texttt{fr}),\ \texttt{go},\ (\texttt{l}, \texttt{fl}, \neg\texttt{r}, \texttt{f}, \texttt{fr}), 1, 3, 1/3\big],\\
O_{T1} &= \big[(\texttt{l}, \texttt{fl}, \neg\texttt{r}, \texttt{f}, \texttt{fr}),\ \texttt{turn},\ (\texttt{l}, \texttt{fl}, \neg\texttt{r}, \neg\texttt{f}, \neg\texttt{fr}), 1, 1, 1/3\big],\\
O_{G2} &= \big[(\texttt{l}, \texttt{fl}, \neg\texttt{r}, \neg\texttt{f}, \neg\texttt{fr}),\ \texttt{go},\ (\texttt{l}, \texttt{fl}, \neg\texttt{r}, \neg\texttt{f}, \neg\texttt{fr}), 1, 3, 1/4\big],\\
O_{G3} &= \big[(\texttt{l}, \texttt{fl}, \neg\texttt{r}, \neg\texttt{f}, \neg\texttt{fr}),\ \texttt{go},\ (\texttt{l}, \texttt{fl}, \neg\texttt{r}), 1, 3, 1/4\big].
\end{aligned}$$

For LOPE, the new generalized operator O_{G3} predicts what the f nal situation will be after applying the action go to the initial situation $(\texttt{l}, \texttt{fl}, \neg\texttt{r}, \neg\texttt{f}, \neg\texttt{fr})$ as well as O_{G1} and O_{G2} do. This is why $P_{O_{G3}} = 1$. When it observes o_4, which is a *confir ation* of operators O_{G1} and O_{G3}, it rewards all operators equal to the observation (O_{G1} and O_{G3}), and punishes all operators that are similar (O_{G1}, O_{G2}, and O_{G3}). Since it rewards both operators O_{G1} and O_{G2}, the K of all operators gets increased to 5. If it would not be increased in one for each equal operator, the sum of quotients P/K for similar operators would not be one, and it would not fulf ll the requirement of being formally sound. The f nal set of operators is:

$$\begin{aligned}
O_{G1} &= \big[(\texttt{l}, \texttt{fl}, \neg\texttt{r}, \neg\texttt{f}, \neg\texttt{fr}),\ \texttt{go},\ (\texttt{l}, \texttt{fl}, \neg\texttt{r}, \texttt{f}, \texttt{fr}), 2, 5, 1/3\big],\\
O_{T1} &= \big[(\texttt{l}, \texttt{fl}, \neg\texttt{r}, \texttt{f}, \texttt{fr}),\ \texttt{turn},\ (\texttt{l}, \texttt{fl}, \neg\texttt{r}, \neg\texttt{f}, \neg\texttt{fr}), 1, 1, 1/3\big],\\
O_{G2} &= \big[(\texttt{l}, \texttt{fl}, \neg\texttt{r}, \neg\texttt{f}, \neg\texttt{fr}),\ \texttt{go},\ (\texttt{l}, \texttt{fl}, \neg\texttt{r}, \neg\texttt{f}, \neg\texttt{fr}), 1, 5, 1/4\big],\\
O_{G3} &= \big[(\texttt{l}, \texttt{fl}, \neg\texttt{r}, \neg\texttt{f}, \neg\texttt{fr}),\ \texttt{go},\ (\texttt{l}, \texttt{fl}, \neg\texttt{r}), 2, 5, 1/4\big].
\end{aligned}$$

4.4. HEURISTIC GENERALIZATION OF OPERATORS

The heuristic generalization of operators is based on the heuristics def ned in (Hayes-Roth, 1983) and (Salzberg, 1985). Hayes-Roth proposed a set of heuristics for revising a faulty (buggy) theory, in the framework of theory revision. Salzberg heuristics are used to correct prediction violations. The next two subsections describe those heuristics in relation to the proposed learning mechanism. The generalization algorithm just applies in a sequence the heuristics described below to the observation, producing a set of new generalized observations, $\mathcal{M}$, whose elements have the following structure:

$$[C, A, F],$$

where C is the generalized initial situation, A is the action, and F is the generalized predicted situation. For the following discussion, the new observation is described by (S_i, A, S_f), the domain operator is described by $[C, A, F, P, K, U]$, and the new mutated observation (m) is $[C_m, A_m, F_m]$. In order to apply a heuristic, there had to be a fault in using the corresponding operator in the observed initial and resulting situation.

4.5. HAYES-ROTH HEURISTICS

Hayes-Roth proposed a superset of the following heuristics for generating new generalized observations from the new observation and the previous set of domain operators. In the case of the application to our robot model, we selected which heuristics were applicable for the chosen representation, and transformed those for correcting violated expectations of plans. In other domains and representations, the heuristics will have to be revisited for validity or create new ones.

– **Retraction:** it generalizes an operator predicted situation so that it is consistent with the new observation. If $F \not\subseteq S_f$, then $m = [C, A, F']$, where F' is a generalization of F and S_f. The algorithm for generalizing is similar to the one used by Mitchell in the version spaces method (Mitchell, 1977). For instance, if $F = (\texttt{l}, \texttt{r}, \neg\texttt{fr})$ and $S_f = (\texttt{l}, \texttt{r}, \texttt{fr}, \texttt{f}, \neg\texttt{fl})$, then F' could be $(\texttt{l},\texttt{r})$.
– **Exclusion:** it restricts the conditions of the operator, so that it does not apply in the observed situation again. Given that $C \subseteq S_i$ and $F \not\subseteq S_f$, then $m = [C', A, F]$, where C' is built by selecting a proposition that does not belong to C (does not matter) and change it to the negation of what appears in the observation. For instance, if $C = (\texttt{l}, \texttt{r}, \neg\texttt{fr})$ and $S_i = (\texttt{l}, \texttt{r}, \neg\texttt{fr}, \texttt{f}, \neg\texttt{fl})$, C' could be $(\texttt{l},\texttt{r},\neg\texttt{fr},\texttt{fl})$.
– **Inclusion:** it generalizes the operator conditions, so that it will later apply in the observed situation. If $F \subseteq S_f$ and $C \not\subseteq S_i$, then $m = [C', A, F]$, where C' is a generalization of C and S_i. For instance, if $C = (\texttt{l}, \texttt{r}, \neg\texttt{fr})$ and $S_i = (\texttt{l}, \texttt{r}, \texttt{fr}, \texttt{f}, \neg\texttt{fl})$, then C' could be $(\texttt{l},\texttt{r})$.

4.6. SALZBERG HEURISTICS

Salzberg proposed a superset of the following heuristics for revising predicting rules in a racing domain, and we also transformed those heuristics to the representation that we used for observations situations and operator conditions and effects.

As in the previous case, we did not implement all heuristics, given that some of them do not have an equivalent when one does not have a knowledge-rich domain theory as Salzberg had.

- **Inusuality:** it restricts the condition of an operator, so that it will not longer apply to the observed initial situation. If $C \subseteq S_i$ and $F \not\subseteq S_f$, then $m = [C', A, F]$, where C' is a specialization of C, adding **all** propositions not appearing in C (does not matter) by the negation of their value in S_i. It differs from Hayes-Roth exclusion, in that it adds *all* propositions to C. For instance, if $C = (\mathtt{l}, \mathtt{r}, \neg\mathtt{fr})$ and $S_i = (\mathtt{l}, \mathtt{r}, \neg\mathtt{fr}, \mathtt{f}, \neg\mathtt{fl})$, C' would be $(\mathtt{l}, \mathtt{r}, \neg\mathtt{fr}, \neg\mathtt{f}, \mathtt{fl})$.
- **Conservationism:** it is a meta-heuristic that selects the generalization heuristic (from the Salzberg ones), that proposes less modif cations in the conditions of an operator.
- **Simplicity:** it is a generalization of the Hayes-Roth retraction heuristic in that it generalizes several operators into one. If several operators have the same conditions and actions, but differ in the predicted situation, then a new generalized observation is generated in which the predicted situation is the generalization of the predicted situations of the operators. For instance, if $F_1 = (\mathtt{l}, \mathtt{r}, \neg\mathtt{fr}, \mathtt{f})$, $F_2 = (\neg\mathtt{l}, \mathtt{r}, \neg\mathtt{fr})$, and $F_3 = (\mathtt{l}, \mathtt{r}, \neg\mathtt{fr}, \neg\mathtt{f})$ are the predicted effects of three operators that share the same action and conditions, then $F' = (\mathtt{r}, \neg\mathtt{fr})$.
- **Adjustment:** when the P/K ratio of an operator falls below a given threshold, it is very unlike that the operator will correctly predict any situation. If it is a generalization of a set of operators (for instance, by application of the simplicity heuristic), this heuristic generates other combinations of those operators that will increase the ratio.

5. Planning

The planner builds a sequence of planning operators[*] that allows to eff ciently reach the top-level goal. This goal is implicit in the system and depends on the measure of utility. At each planning cycle, the system receives the highest utility (as, for instance, being on top of an energy point, or, in the case of robotic soccer, scoring a goal) if it reaches a situation yielding such utility. In case another domain requires a more classical high-level set of goals (as in the case of the blocksworld or logistics transportation), a richer representation would be needed. The planning and learning components would have to be changed accordingly, but the overall architecture and techniques would still be valid.

Since each operator has its own utility, the planner will try to build plans that transform each current situation into the situation described by the condition of

[*] In this case, the term operator and action are equivalent with respect to planning and execution, given that operators do not have variables as in the case of classical planning (Fikes et al., 1972).

the operator with the highest utility; that is, the operator that achieves the highest utility situation (its f nal situation). Therefore, these conditions are subgoals of the top-level goal. The resulting plan will allow the execution module to perform a set of actions that interact with the environment in such a way that it arrives to a situation in which the action of the highest utility operator can be executed (the conditions are met), thus achieving that level of utility.

5.1. BUILDING A PLAN

The planning algorithm proceeds as follows (also shown in Figure 8). At the beginning, there are no operators, so the system generates a default plan using the algorithm shown in Figure 9. Default planning randomly selects whether to act randomly, or to act by approaching a close obstacle. Since the planner should return a list of operators, and `default-plan` only selects actions randomly, this function creates for each action in the returned plan a dummy operator.

If the system already has some operators, the function `generate-goals` builds a list of goals, each of them is a pair (*situation,operator*), where *situation* is the f nal situation of the *operator*. This list is ordered by decreasing values of the utility of their respective operators. For each subgoal, the planner tries to f nd a plan that can

```
Function Plan ($S$, $\mathcal{O}$, $\mathcal{A}$): $P$

$S$: Current situation (8-bit vector)
$\mathcal{O}$: Set of operators
$\mathcal{A}$: Set of actions
$P$: Plan (list of operators)
$G$: List of pairs (situation,operator)

If $\mathcal{O} = \emptyset$
Then $P$ :=default-plan($S$, $\mathcal{A}$)
Else $G$:=generate-goals($\mathcal{O}$)
    $P := \emptyset$
    While $G \neq \emptyset$ AND $P = \emptyset$ do
        $g$:=pop($G$)
        $P$:=build-plan($S$, $g$, $\mathcal{O}$)
    If $P = \emptyset$
    Then $P$ :=default-plan($S$, $\mathcal{A}$)
Return $P$
```

Figure 8. Algorithm that obtains a plan to be executed in the environment.

```
Function Default-plan ($S$, $\mathcal{A}$): $P$

$S$: Current situation (8-bit vector)
$\mathcal{A}$: Set of possible actions
$A$: List of actions that conform the plan
$P$: Plan (list of operators)

$r$:=random(0,1)
If $r \geq 0.5$
Then $A$:={random-selection($\mathcal{A}$)}
Else $A$:=goto-closest-obstacle($S$, $\mathcal{A}$)
Forall $a \in A$ do
    $P$:=$P \cup \{[S, a, ?, ?, ?, ?]\}$
    $S$:=?
Return $P$
```

Figure 9. Algorithm that selects a random action or goes to the closest obstacle.

transform the current situation S into one of the subgoals (function build-plan explained below). Since it first tries the goals with higher utilities, and it stops when it finds a plan, the planner will find a plan for achieving the *highest utility reachable goal*. If the planner cannot find a plan for any goal, it generates a default plan according to the default planning procedure described above.

The function build-plan creates a graph by backward chaining on the goal. Since the goals are pairs (*situation,operator*), the root of the search tree will be the *situation*, that will only have one successor labeled with the *operator* of the goal. For each situation in the search tree, it creates a node, and a successor for each operator whose final situation matches that situation, and continues backwards until it cannot expand more nodes. Goal loops (repetition of the same situation in the path from a node to the root) are detected and search stops under those nodes. When it finishes the expansion of the tree, if the current situation appears in the graph, there exists at least one plan that can achieve the goal from the current situation. This algorithm can be easily changed to find the first plan, instead of exploring the whole search tree. In the reported experiments, we found that for that domain it was not needed to stop before completing the whole search tree.

5.2. EXAMPLE OF PLANNING EPISODE

As an example of how this algorithm proceeds, suppose that the system already built the following set of operators:

$$O_1 = (S_1, A_1, S_2, 3, 4, 0.6),$$

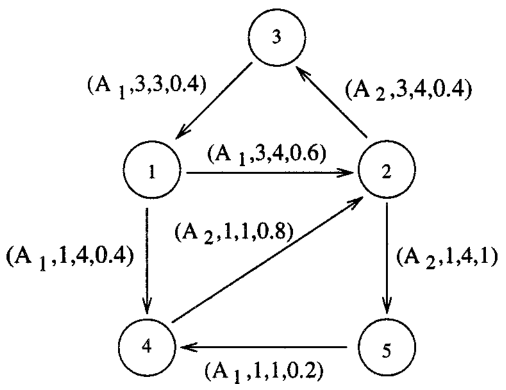

Figure 10. Example of graph of planning operators.

$$\begin{aligned}
O_2 &= (S_1, A_1, S_4, 1, 4, 0.4),\\
O_3 &= (S_2, A_2, S_3, 3, 4, 0.4),\\
O_4 &= (S_2, A_2, S_5, 1, 4, 1),\\
O_5 &= (S_3, A_1, S_1, 3, 3, 0.4),\\
O_6 &= (S_4, A_2, S_2, 1, 1, 0.8),\\
O_7 &= (S_5, A_1, S_4, 1, 1, 0.2).
\end{aligned}$$

where there are two actions A_1 and A_2, and f ve situations S_1 to S_5.* The search space corresponding to those operators is shown in Figure 10, where each node represents a situation, and each arc is labeled with a tuple (A, P, K, U), where A is the action of the operator, that transforms a situation into another, P and K are the features that capture the information on success probability (explained below), and U is the operator utility.

Given that search space, if the planner tries to f nd a plan to achieve the higher utility reachable goal from the current situation S_1, it would f rst generate the following list of pairs (*situation,operator*) in descendent order of utility:

$$[(S_5, O_4)(S_2, O_6)(S_2, O_1)(S_4, O_2)(S_3, O_3)(S_1, O_5)(S_4, O_7)].$$

As (S_5, O_4) has the highest utility level, the system builds the search tree shown in Figure 11, where the root is the situation S_5, its only successor is the condition part of the operator O_4, situation S_2, and the arcs are labeled with the actions and operators that transform a situation into another. Search stops under nodes of situations S_3 (twice) and S_5, since the only way to obtain those situations is from situation S_2 which would cause a goal loop in both cases.

* In this example, we use situations instead of conditions to simplify the explanation. One can think of situations as instantiated conditions.

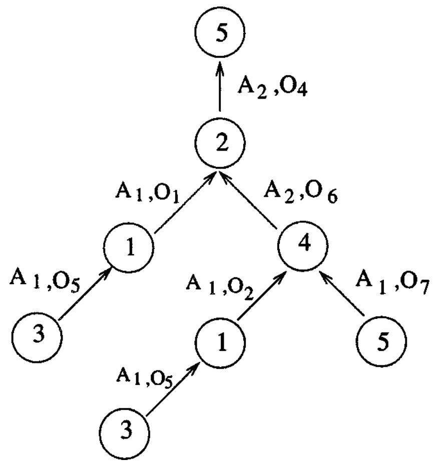

Figure 11. Search tree generated when planning to achieve situation S_5 from S_1.

There are two plans that reach S_5 (auto generated goal) from S_1 (current situation): $O_1 \circ O_4$* (actions A_1 and A_2) and $O_2 \circ O_6 \circ O_4$ (actions A_1, A_2, and A_2). The planner selects the shortest plan, which is $O_1 \circ O_4$ ($A_1 \circ A_2$).

5.3. STOCHASTIC PLANNING

In order to estimate the probability of success of plans, the planner is based on an extension of the theory of stochastic automata. The knowledge that the system has at a given time, the set of planning operators, can be viewed as a model of how its environment will react to the system's actions. The quotient P_{O_i}/K_{O_i} of a given operator O_i, is the probability estimator of the fact that given the action A_{O_i} applied to a situation S_j that matches the operator conditions ($C_{O_i} \subseteq S_j$) results in a situation S_k that verif es the predicted effects of the operator ($F_{O_i} \subseteq S_k$). In (García-Martínez, 1997), it is shown that this estimator is an unbiased estimator that follows a multinomial probability distribution. Therefore, the knowledge that the system has about the effects of an action A_i at a given instant can be represented by the transition matrix M_{A_i}, that has, in the (j, k) position, the quotient P_O/K_O of operators O whose action is A_i, its conditions are S_j, and the predicted effects S_k (Calistri-Yeh, 1990).

The Ps and Ks of the plan operators can be used in the evaluation of the plans that are generated. This "a priori" estimation of the plan success probability allows to discard plans with a low probability of success ($P/K < \tau$), where τ is a predef ned threshold. This property is critical when the system must act without supervision.

* $\circ$ represents the composition of actions.

As an example, in the previous plan, the transition matrix M_{A_1} associated to the action A_1, the transition matrix M_{A_2} associated to the action A_2, and the transition matrix M_P of the plan $P = A_1 \circ A_2$ are:

$$M_{A_1} = \begin{array}{c|ccccc|} & S_1 & S_2 & S_3 & S_4 & S_5 \\ \hline S_1 & 0 & \frac{3}{4} & 0 & \frac{1}{4} & 0 \\ S_2 & 0 & 0 & 0 & 0 & 0 \\ S_3 & 1 & 0 & 0 & 0 & 0 \\ S_4 & 0 & 0 & 0 & 0 & 0 \\ S_5 & 0 & 0 & 0 & 1 & 0 \\ \hline \end{array}$$

$$M_{A_2} = \begin{array}{c|ccccc|} & S_1 & S_2 & S_3 & S_4 & S_5 \\ \hline S_1 & 0 & 0 & 0 & 0 & 0 \\ S_2 & 0 & 0 & \frac{3}{4} & 0 & \frac{1}{4} \\ S_3 & 0 & 0 & 0 & 0 & 0 \\ S_4 & 0 & 1 & 0 & 0 & 0 \\ S_5 & 0 & 0 & 0 & 0 & 0 \\ \hline \end{array}$$

$$M_P = M_{A_1} \times M_{A_2} = \begin{array}{c|ccccc|} & S_1 & S_2 & S_3 & S_4 & S_5 \\ \hline S_1 & 0 & \frac{1}{4} & \frac{9}{16} & 0 & \frac{3}{16} \\ S_2 & 0 & 0 & 0 & 0 & 0 \\ S_3 & 0 & 0 & 0 & 0 & 0 \\ S_4 & 0 & 0 & 0 & 0 & 0 \\ S_5 & 0 & 1 & 0 & 0 & 0 \\ \hline \end{array}$$

From the analysis of M_P, the probability that the plan P applied to the situation S_1 achieves the situation S_2 is $\frac{1}{4}$, the probability that the plan P applied to the situation S_1 achieves the situation S_3 is $\frac{9}{16}$, and the probability that the plan P applied to the situation S_1 achieves the situation S_5 is $\frac{3}{16}$.

6. Learning by Sharing

After performing experiments that conf rmed that the overall architecture performed very well, as described in Section 7, we decided to experiment with the inclusion of new agents of the same type (García-Martínez and Borrajo, 1998). These agents learn and share what they learn in the same grid conf guration. Agents cannot occupy the same position in the grid, and the sensors of one agent consider the other agents as obstacles. Under this framework, each agent continuously learns, plans and executes. However, when they were close to another agent, they were allowed to communicate in order to interchange what they learned, operator descriptions.

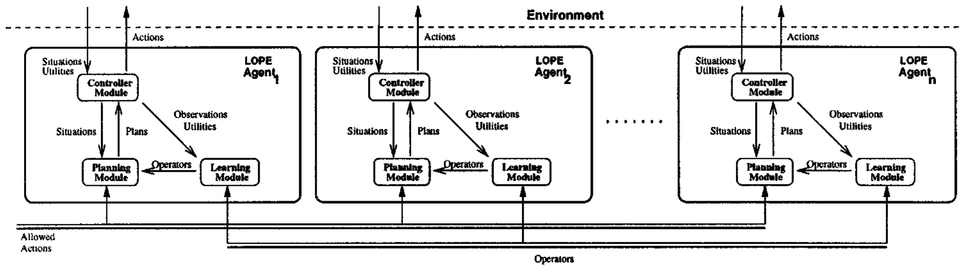

Figure 12. Architecture of a group of LOPE agents.

Function `Complete-sharing` ($\mathcal{O}_1$, $\mathcal{O}_2$): $\mathcal{O}_1$

$\mathcal{O}_1$: Set of operators of agent 1
$\mathcal{O}_2$: Set of operators of agent 2

```
Forall $O_i \in \mathcal{O}_2$ do
    If exists $O_j \in \mathcal{O}_1$ such that similar-op($O_j$, $O_i$)
    Then punish-similar-operators($O_i$, $\mathcal{O}_1$, $K_{O_i}$)
        If exists $O_k \in \mathcal{O}_1$ such that equal-op($O_i$, $O_k$)
        Then reward-operator($O_k$, $P_{O_i}$)
        Else $K_{O_i} := K_{O_j}$
            $\mathcal{O}_1 := \mathcal{O}_1 \cup \{O_i\}$
    Else $\mathcal{O}_1 := \mathcal{O}_1 \cup \{O_i\}$
Return $\mathcal{O}_1$
```

Figure 13. Algorithm that integrates the operators of two agents.

Figure 12 shows a schematic view of the architecture, where there can be n LOPE agents. Each of the agents receives as input: perceptions from the environment (situations and utilities); set of actions that it can perform; and operators learned by other agents. The output of each agent is a sequence of actions over time (for the environment), and, regularly, the set of operators that it learned (for the other agents).

We devised two types of knowledge sharing strategies:

– *Complete sharing.* Every pair of agents integrates their respective theories (set of operators) using all operators in the sets. The algorithm is shown in Figure 13. For each operator of another agent (a_2), an agent (a_1) looks for similar operators in its theory. If there is no such similar operator, then the a_2's operator is included in the set of operators of a_1. If a similar operator is found, then all such operators are punished with the K of the operator of a_2. Then, if an equal operator of a_1 exists, it is rewarded with the P of a_2's operator. If there is no equal operator, then it is included in a_1's operators with the K of its similar operators in a_1's theory.

– *Most reliable operator sharing*. Every time two agents share their knowledge, only the most liable operators are shared (the ones that maximize the quotient P/K). The only difference with the prior algorithm is that instead of providing it as input with $\mathcal{O}_2$, the algorithm is called with the set of most liable operators. This set is computed by selecting from each set of similar operators of an agent, the one with maximum P/K.

7. Experiments and Results

We tested LOPE in different environments. In each environment, it learns models that predict the effects of applying actions to situations. This allows LOPE to plan for achieving its goals: arriving at points where batteries can be charged (energy points).

The general setting for the experiments can be described as follows:

1. Generate 50 experiments by:
   (a) Randomly choosing an environment (grid). The sizes of the grids were 700 × 1000 pixels. Each pixel could be empty, or have an obstacle, an energy point, or a robot. In the f rst three experiments reported, we used a 10%–20% randomly chosen occupancy of the grid by obstacles. In the last one, we varied the percentage of obstacles in the environment to see how that affected its performance. Also, energy points were placed randomly (their number being randomly chosen between 10 and 20);
   (b) Randomly choosing an initial position of the robot in the chosen environment;
   (c) Executing LOPE for 8,000 cycles. One cycle means the pair perceive/act; and
   (d) Recording the state of the main variables every 500 cycles
2. Compute the average of each variable recorded (every 500 cycles) with the 50 experiments

We performed four experiments to test the behavior of LOPE: performance of the learning and planning system in a single environment; generalization over different environments; knowledge sharing among different agents; and importance of the percentage of obstacles in the environment.

7.1. RESULTS ON LEARNING AND PLANNING

In the f rst experiment, our goal was to test f rst the performance of the system. We compared four versions of the system:

– The base planner, in which operators are created directly from the observations, following Fritz et al. work (Fritz et al., 1989). This conf guration

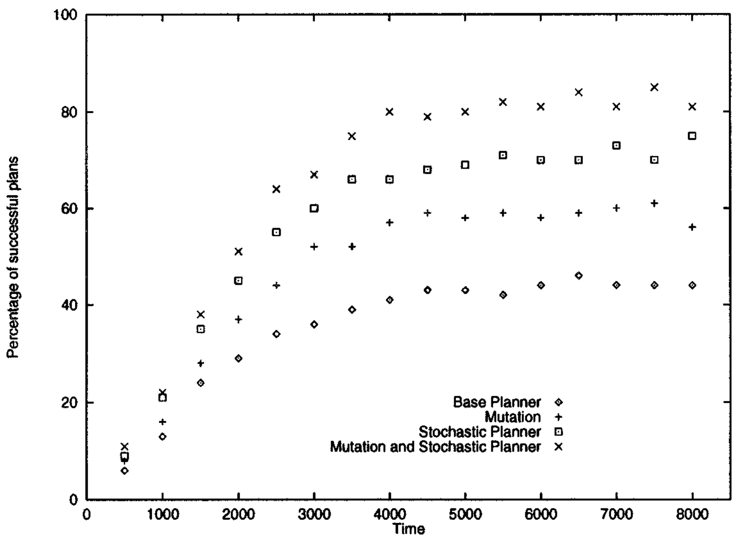

Figure 14. Results of comparing four versions of the system: the base planner, the base planner with generalization applied to operators, the base planner with probabilities estimations associated to operators, and the base planner with mutated operators and probabilities estimations.

is better than the simplest ones, which would act randomly, or by curiosity (always approaching the closest obstacle).

– The base problem solver using operators learned using heuristic generalization (García-Martínez, 1993).

– The base problem solver estimating for each operator its probability of success (García-Martínez and Borrajo, 1996).

– The base problem solver, in which operators are mutated, and a probability estimator is assigned to each operator (García-Martínez, 1997).

We used the percentage of successful plans when comparing these four versions of the system, and the results of the experiment are shown in Figure 14. These results clearly show that the combination of generalization and probability estimation outperform the base planner behavior, and, also, the separate use of any of them. The combined use of generalization and probabilities make the system converge towards a 80% of success plans, while the base planner converges towards around 40%. We believe this is caused by the cooperation between: the generalization effect that the heuristic generalization has; and the incorporation of the weighting mechanism for planning that allows to specify the plans acceptability, increasing the number of successful plans.

Figure 15 shows how the number of observed situations and generated operators evolved during learning.

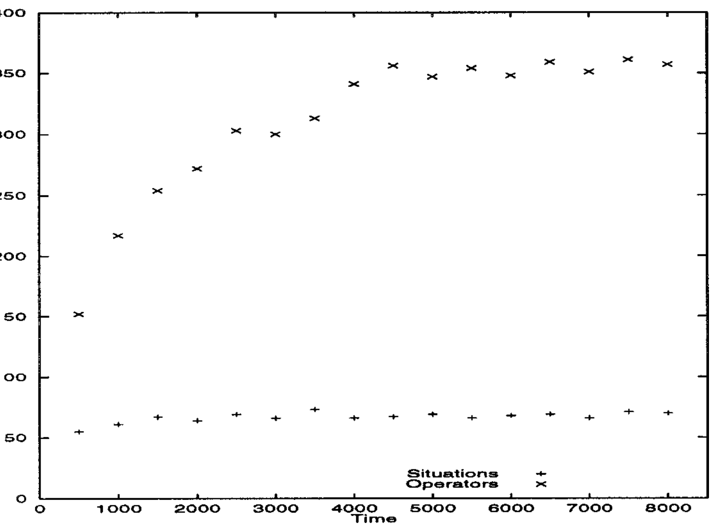

Figure 15. Evolution of the number of situations and operators with respect to time.

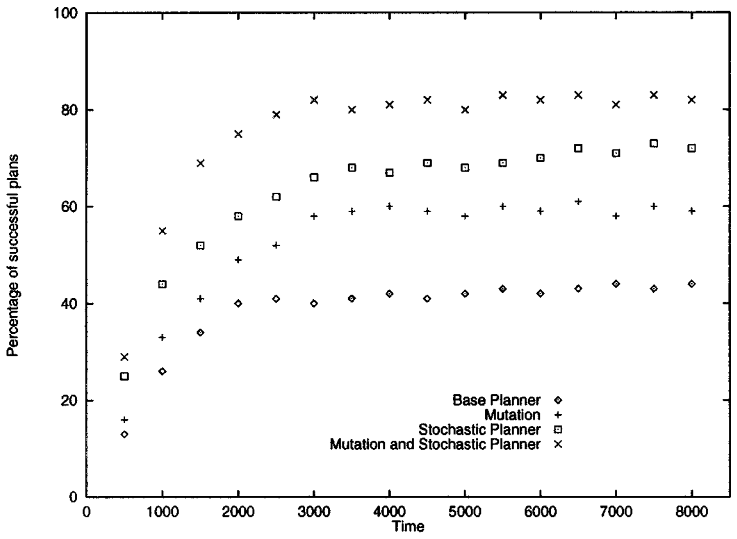

Figure 16. Results of comparing four versions of the system with respect to learning convergence.

7.2. RESULTS ON GENERALIZATION

The second experiment was performed to show that the introduction of initial knowledge to the system improved the behaviour with respect to learning convergence, while leaving intact the relative differences between the four conf gurations. We randomly generated a set of environments, $\mathcal{E}$, and averaged the results of running 50 times the following experiment: an environment e was randomly selected from $\mathcal{E}$; 8000 cycles were run on e; another environment $e' \neq e$ was chosen from

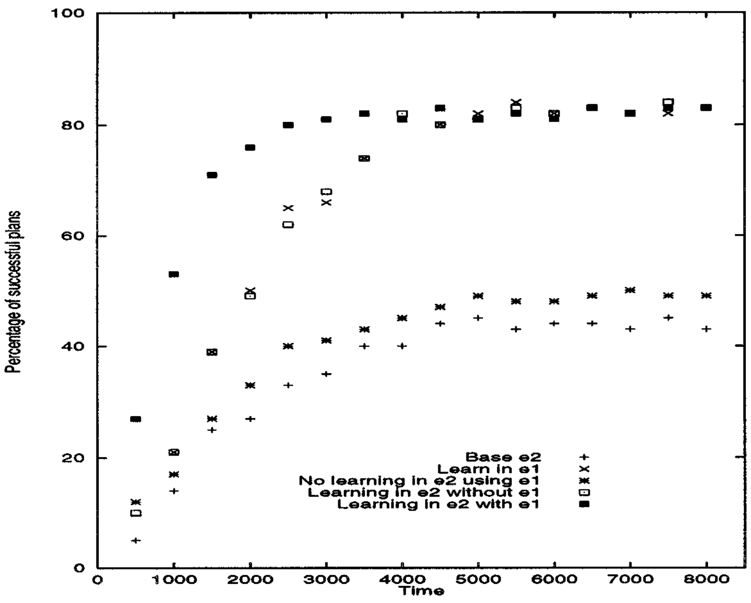

Figure 17. Results of comparing the convergence on two different grids.

the set $\mathcal{E}$; 8000 cycles were run on e', using the learned operators in e; and results were collected. The results are shown in Figure 16 where it can be observed that the use of previously learned knowledge, even in another environment, continues improving the behavior of the system with learning and stochastic planning. Also, and more importantly, the convergence towards a very good percentage of successful plans improved, with respect to the results in Figure 14. For instance, while, in the f rst experiment, the 70% of successful plans was achieved at around 3200 cycles, in the second experiment, the same success ratio was achieved at 1500 cycles.

This phenomenon can be better seen in Figure 17, where the convergence effect is compared. It shows the comparison among: the base planner in the second grid e_2 without initial knowledge; learning in e_1 without initial knowledge; the base planner without learning in e_2 using as initial knowledge the learned operators in e_1; learning in e_2 without initial knowledge; and learning in e_2 using the learned operators in e_1 as initial knowledge. As it can be seen, if the base planner does not learn, using the initial knowledge from another grid, improves the behaviour, but not as much as learning also in the second grid. If it learns using that initial knowledge, the convergence of the learning phase improves radically. Also, as it should be expected, the learning rates in both grids without prior knowledge are almost identical, given that each point in the graphic has been generated from an average of 50 experiments.

We also performed a similar experiment to the previous one in which a third grid was introduced to test further knowledge transfer and learning convergence. In this case, we compared the behaviour in a third grid e_3 of learning in that grid using

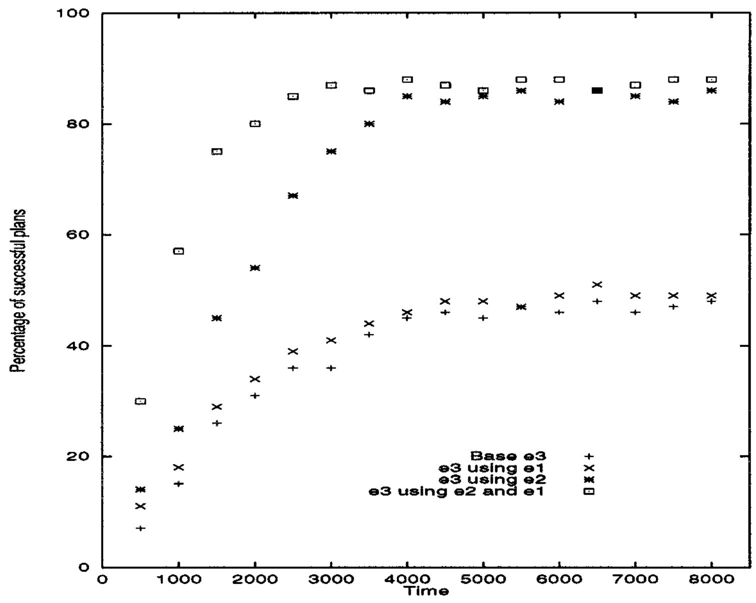

Figure 18. Results of comparing convergence on three different grids.

previously learnt knowledge in e_1 and e_2 (e_3 using e_2 and e_1), with the behaviour in e_3 without learning of: no previous knowledge (base e_3); knowledge learned in e_1 and refned in e_2 only through changing probabilities and not learning new operators (e_3 using e_1); knowledge learned in e_1, refned both by learning new operators and probabilities through its use in e_2 (e_3 using e_2).

As it can be seen, the best behaviour is obtained by learning in each grid, using what was learned in the previous ones. Also, learning new operators and probabilities in e_2 using what was learned in e_1 is clearly better (when used in e_3) than not learning new operators in e_2 using what was learned in e_1. This is due to the fact that observations made in e_1 are rarely seen again in e_2, and their probabilities (P/K) will be reduced while learning probabilities in e_2. This causes that plans will have very low success probability in e_3, and will be scarcely used in that grid.

7.3. RESULTS ON SHARING KNOWLEDGE

In order to test the effect of sharing the knowledge among the agents, we performed new experiments which we then compared with a summary of the best results of the previous ones. In the multiple agents setup, the agents shared their knowledge when they were 20 pixels away from another. We compare here six experiments:

– BP: the base planner as explained before.
– SS: a single LOPE agent learning in a single grid, in which operators are generalized, and a probability estimator is assigned to each operator (García-

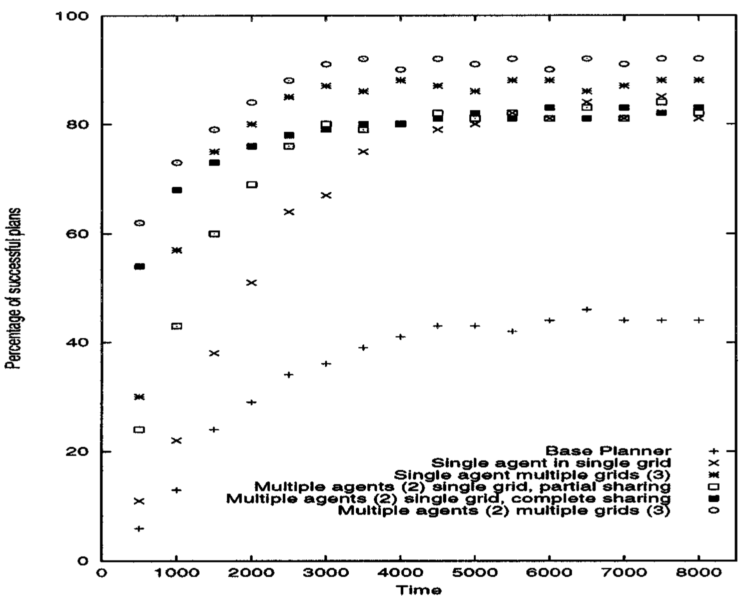

Figure 19. Results of comparing different versions of the system with respect to sharing knowledge among several agents.

Martínez and Borrajo, 1997). This estimator is the quotient P/K of each learned operator. Also, it is used to assign a conf dence to the generated plans, so that plans with low conf dence are discarded.*

- SM: a single LOPE agent that learns in a grid conf guration g_1, ref nes its knowledge both by learning new operators and probabilities through its use in another conf guration g_2, and using this prior knowledge to learn in a third grid g_3 as described in the previous experiment.
- MSP: a set of LOPE agents (we used two for these experiments) learning at the same time in the same grid conf guration with the partial sharing strategy.
- MSC: a set of LOPE agents learning at the same time in the same grid conf guration with the complete sharing strategy.
- MM: a set of LOPE agents learning at the same time in three different grid conf gurations in the same way as the experiment of a single agent in three different grids.

We used the percentage of successful plans when comparing these versions of the system, and the results of the experiment are shown in Figure 19. First, these results clearly show that the combination of generalization and probability estima-

* The decisions of the agent are based on sensory input only when there is no plan on execution. We have shown previously that the P/K of similar operators follows a multinomial distribution of probability and that is an unbiased estimator of the probability. Also, when an exact theory of the domain exists, the operators that have been built applying the learning mechanism based on observations converge to the exact ones.

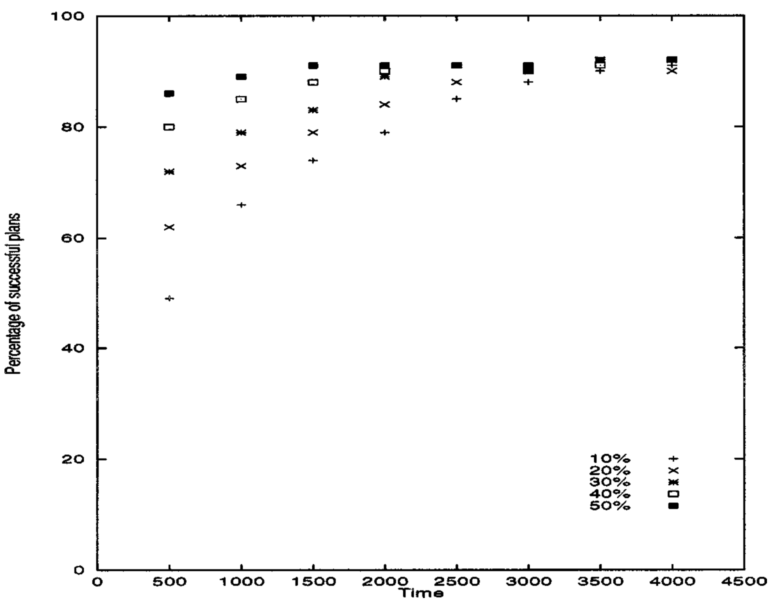

Figure 20. Results of comparing different environment types.

tion outperforms the base planner behavior even in the simplest case: a single agent learning in a single conf guration (SS case). The combined use of generalization and probabilities makes the system converge towards a 80% of success plans, while the base planner converges towards around 40%.

Using the initial knowledge from another grid, radically improves the behavior and convergence of a single agent (SM case). This case turns to behave better than using two agents in a single grid with partial sharing (MSP case). However, using two agents in a single grid with complete sharing (MSC) improves in convergence rate with respect to the SM case, but it is worse in the long run, since it converges to a lower percentage rate than the SM case. It means that sharing knowledge among agents at the beginning is better than using different grids, but, at the end, using more than one grid is better. Finally, the multiple agents multiple grids case (MM) outperforms all other conf gurations in convergence and in f nal percentage rate. See, for instance, that the percentage rate after 500 cycles is above 60% (62%), while in the BP case is around 10%, and the second best convergence rate is around 54%. Also, it is the only experiment that reaches a percentage rate of above 90%.

One aspect that would have to be better studied when dealing with real robots is the fact that the sensors of two robots are never exactly equal, so the representations developed by one robot may not have the same effects on another robot. Since our model accounts for a probability of success for each operator that would allow to reduce the impact of those different mappings from sensors into representations.

7.4. RESULTS ON VARYING THE PERCENTAGE OF OBSTACLES

In order to see how the percentage of obstacles in the domain affected the system, we performed a last experiment in which we varied this percentage from 10% up to 50%. The results are shown in Figure 20. We used multiple agents, learning in multiple grids, that interchange all their models everytime they communicate. As it can be seen, this model performs better in dense environments than in sparse environments. This can explained by the fact that dense environments provide more information with respect to different types of sensory inputs, which allows to learn better models.

8. Related Work

Currently, the most related work is the one on reinforcement learning techniques within the Markov Decision Processes (MDP) paradigm (Mahavedan and Connell, 1992; Sutton, 1990; Watkins and Dayan, 1992). Also, current techniques that deal with Partially Observable Markov Decision Processes (POMDP) are very close to this approach (Cassandra et al., 1994). Usually, they integrate reinforcement learning, planning and execution based on approximated dynamic programming. There are some differences between this type of approaches and our approach:

– **Representation of action descriptions.** In the case of classical reinforcement learning techniques, there is no representation of the actions, since they are not used for explicit planning. It is only needed to keep the action name, and the procedure for executing each action in the environment. However, in our case, since we perform an explicit backward search over the operator descriptions, we need an explicit representation on how those actions change the environment.

– **Global vs. local reinforcement.** The reinforcement procedure for most techniques is local to an operator (the term operator is understood here as the connection between two states through the execution of an action). While, in our case, the reinforcement of an operator explicitly implies the punishment of similar ones, so there is a global reinforcement of the same action.

– **Representation of states.** We use *symbolically generalized states*, instead of instantiated states as most other work in reinforcement learning, or non-symbolically based generalized states (such as neural networks (Lin, 1993) or mechanisms based on vector quantization (Fernández and Borrajo, 1999)). In fact, our approach can be viewed as a method for producing a virtual generalized Q table using global reinforcement. Similar approaches, group sets of similar states and/or actions on big state/action spaces (Boutilier et al., 1995; Dean and Givan, 1997; Fernández and Borrajo, 1999). Most of this work uses different representation schemas, such as belief networks.

– **Type of planning scheme.** While reinforcement learning has been usually applied for reactive planning (with some exceptions), our approach lies closer to the classical planning approach (plans are generated in a search-based fashion and later monitored for divergences between predicted and observed states) (García-Martínez and Borrajo, 1997).

– **Handling of the temporal credit assignment problem.** For classical reinforcement learning techniques, the concept of time delayed reward is very important. Therefore, their algorithms concentrate on how to assign credit to actions whose reward is only known after some time has passed. In our approach, each learning episode is handled independently of what happened before.

Within the classical reinforcement learning framework, the work by Tan (Tan, 1993) could be considered a predecessor of our work. He explores the cooperation among agents by sharing instantaneous information (perceptions, actions or rewards), sequences of perception-action-reward, and learned policies.

Other approaches that integrate planning, learning and execution are:

– OBSERVER (Wang, 1996) integrates planning and learning. Wang proposes an incremental approach for operators revision, where operators evolve during the execution of the system. However, there is no memory of past versions of the operators as in LOPE. Another difference relies in the representation language for operators. Her work used the representation language of PRODIGY4.0 operators (Veloso et al., 1995) that is based on predicate logic, since its goal is to perform classical high-level planning. Our approach uses a representation that is closer to the inputs and outputs of a more reactive (robotic) system, with low-level planning.

– The GINKO system (Barbehenn and Hutchinson, 1991), the LIVE system (Shen, 1993), and the work of (Safra and Tennenholtz, 1994). They differ from the proposed architecture in the fact that they do not take into account reinforcement nor heuristic-based ref nement of operators.

– Christiansen (Christiansen, 1992) also addresses the problem of learning operators (task theories) in a robotic domain. However, in his work there is no revision process as our heuristic-based ref nement process.

– Other systems for robotic tasks are (Bennet and DeJong, 1996) and (Klingspor et al., 1996). The f rst one deals with the concept of *permissiveness*, that def nes qualitative behavior for the operators. The second one uses Inductive Logic Programming for learning the operators of the domain by doing a transformation from the sensor data into predicate logic. They both differ from our approach in that they need some type of prior background knowledge, either a predef ned domain theory in the form of initial operators, or external instruction and knowledge on how to perform the transformation.

9. Conclusions

There are many real world problems where there is no domain theory available, the knowledge is incomplete, or it is incorrect. In those domains, autonomous intelligent systems, def ned as systems that learn, self-propose goals, and build plans to achieve them, sometimes are the only alternative to acquire the needed domain description.

In this paper, we have presented an architecture that learns a model of its environment by observing the effects of performing actions on it. The LOPE system autonomously interacts with its environment, self-proposes goals of high utility for the system, and creates operators that predict, with a given probability estimator, the resulting situation of applying an action to another situation. Learning is performed by three integrated techniques: rote learning of an experience (observation) by creating an operator directly from it; heuristic generalization of incorrect learned operators; and a global reinforcement strategy of operators by rewarding and punishing them based on their success in predicting the behavior of the environment.

The results show that the integration of those learning techniques can greatly help an autonomous system to acquire a theory description that models the environment, thus achieving a high percentage of successful plans. The use of knowledge sharing strategies among the agents have shown that sharing the learned knowledge can greatly help an autonomous system to acquire a theory description that models the environment, thus achieving a high percentage of successful plans, and also improving the convergence rate for obtaining a successful theory.

An important issue when allowing sharing of operators among agents, is related to the differences on their sensors, which causes different ways of perceiving the world, and, therefore, different biases towards the generation of operators. We have not yet studied this effect, although one possible way of solving it could be by learning other agents biases, in order to perform a more informed sharing of knowledge.

With respect to the scalability of the approach, we are now performing experiments in a much more complex, noisy, with hidden states, and multi-agent domain, such as the Robosoccer. We believe that through the use of the probabilities estimations, and the heuristic generalisation of operators, we will be able to cope with the complexity of that domain. Another domain we have also shown elsewhere that we can apply this architecture to has been a subset of matrices algebra (García-Martínez, 1997).

Acknowledgements

The authors would like to thank anonymous reviewers for their useful hints about the paper.

References

Ashish, N., Knoblock, C., and Levy, A.: 1997, Information gathering plans with sensing actions, in: S. Steel (ed.), *Proc. of the 4th European Conf. on Planning*, Toulouse, France, pp. 15–27.

Barbehenn, M. and Hutchinson, S.: 1991, An integrated architecture for learning and planning in robotic domains, *Sigart Bulletin* **2**(4), 29–33.

Bennet, S. W. and DeJong, G.: 1996, Real world robotics: Learning to plan for a robust execution, *Machine Learning* **23**(2/3), 121–162.

Borrajo, D. and Veloso, M.: 1997, Lazy incremental learning of control knowledge for eff ciently obtaining quality plans, *AI Rev. J. Special Issue on Lazy Learning* **11**(1–5), 371–405.

Boutilier, C., Dearden, R., and Goldszmidt, M.: 1995, Exploiting structure in policy construction, in: *Proc. of the 14th Internat. Joint Conf. on Artif cial Intelligence (IJCAI-95)*, Montreal, Quebec, Canada, pp. 1104–1111.

Brooks, R. A.: 1986, A roboust layered control system for a mobile robot, *IEEE J. Robotics Automat.* **2**(1), 14–23.

Calistri-Yeh: 1990, Classifying and detecting plan-based misconceptions for robust plan recognition, PhD Thesis, Department of Computer Science, Brown University.

Carbonell, J. G. and Gil, Y.: 1990, Learning by experimentation: The operator ref nement method', in: R. S. Michalski and Y. Kodratoff (eds.), *Machine Learning: An Artif cial Intelligence Approach, Vol. III*, Palo Alto, CA: Morgan Kaufmann, pp. 191–213.

Cassandra, A., Kaebling, L., and Littman, M.: 1994, Acting optimally in partially observable stochastic domains, in: *Proc. of the American Association of Artif cial Intelligence (AAAI-94)*, pp. 1023–1028.

Christiansen, A.: 1992, Automatic acquisition of task theories for robotic manipulation, PhD Thesis, School of Computer Science, Carnegie-Mellon University, Pittsburgh, PA.

Dean, T. and Givan, R.: 1997, Model minimization in Markov decision processes, in: *Proc. of the American Association of Artif cial Intelligence (AAAI-97)*.

Falkenhainer, B.: 1990, A unif ed approach to explanation and theory formation, in: J. Shrager and L. P. (eds), *Computational Models of Scientif c Discovery and Theory Formation*, Morgan Kaufmann.

Fernández, F. and Borrajo, D.: 1999, Vector quantization applied to reinforcement learning, in: M. Veloso (ed.), *Working notes of the IJCAI'99 3rd Internat. Workshop on Robocup*, Stockholm, Sweden, pp. 97–102.

Fikes, R. E., Hart, P. E., and Nilsson, N. J.: 1972, Learning and executing generalized robot plans, *Artif cial Intelligence* **3**, 251–288.

Fikes, R. E. and Nilsson, N. J.: 1971, STRIPS: A new approach to the application of theorem proving to problem solving, *Artif cial Intelligence* **2**, 189–208.

Fritz, W., García-Martínez, R., Blanqué, J., Rama, A., Adobbati, R., and Samo, M.: 1989, The autonomous intelligent system, *Robotics Autonom. Systems* **5**(2), 109–125.

García-Martínez, R. and Borrajo, D.: 1996, Unsupervised machine learning embedded in autonomous intelligent systems, in: *Proc. of the 14th IASTED Internat. Conf. on Applied Informatics*, Innsbruck, Austria, pp. 71–73.

García-Martínez, R. and Borrajo, D.: 1997, Planning, learning, and executing in autonomous systems, in: S. Steel (ed.), *Recent Advances in AI Planning, 4th European Conf. on Planning, ECP'97*, Toulouse, France, pp. 208–220.

García-Martínez, R. and Borrajo, D.: 1998, Learning in unknown environments by knowledge sharing, in: J. Demiris and A. Birk (eds.), *Proc. of the 7th European Workshop on Learning Robots, EWLR'98*, Edinburgh, Scotland, pp. 22–32.

García-Martínez, R.: 1993, Heuristic theory formation as a machine learning method, in: *Proc. of the VI Internat. Symposium on Artif cial Intelligence*, México, pp. 294–298.

García-Martínez, R.: 1997, Un Modelo de aprendizaje por observación en planif cación, PhD Thesis, Facultad de Informática, Universidad Politécnica de Madrid.

Hayes-Roth, F.: 1983, Using proofs and refutations to learn from experience, in: R. S. Michalski, J. G. Carbonell, and T. M. Mitchell (eds.), *Machine Learning, An Artif cial Intelligence Approach*, Palo Alto, CA, Tioga Press, pp. 221–240.

Kitano, H., Asada, M., Kuniyoshi, Y., Noda, I., and Osawa, E.: 1995, RoboCup: The robot world cup initiative, in: *Proc. of the IJCAI-95 Workshop on Entertainment and AI/Life*, pp. 19–24.

Klingspor, V., Morik, K. J., and Rieger, A. D.: 1996, Learning concepts from sensor data of a mobile robot, *Machine Learning* **23**(2/3), 305–000.

Langley, P.: 1983, Learning effective search heuristics, in: *Proc. of the 8th Internat. Joint Conf. on Artif cial Intelligence*, Los Altos, CA, pp. 419–421.

Lin, L.-J.: 1993, 'Scaling-up reinforcement learning for robot control, in: *Proc. of the 10th Internat. Conf. on Machine Learning*, Amherst, MA, pp. 182–189.

Mahavedan, S. and Connell, J.: 1992, Automatic programming of behavior-based robots using reinforcement learning, *Artif cial Intelligence* **55**, 311–365.

Matellán, V., Borrajo, D., and Fernández, C.: 1998, Using ABC^2 in the RoboCup domain, in: H. Kitano (ed.), *RoboCup-97: Robot Soccer World Cup I*, pp. 475–483.

Minton, S.: 1988, *Learning Effective Search Control Knowledge: An Explanation-Based Approach*, Boston, MA, Kluwer Academic, Dordrecht.

Mitchell, T.: 1977, Version spaces: A candidate elimination approach to rule learning, in: *Proc. of the 5th IJCAI*, MIT, Cambridge, MA, pp. 305–310.

Safra, S. and Tennenholtz, M.: 1994, On planning while learning, *J. Artif cial Intell. Res.* **2**, 111–129.

Salzberg, S.: 1985, Heuristics for inductive learning, in: *Proc. of the 9th Internat. Joint Conf. on Artif cial Intelligence*, Los Angeles, CA, pp. 603–609.

Shen, W.: 1993, Discovery as autonomous learning from enviroment, *Machine Learning* **12**, 143–165.

Simmons, R. and Mitchell, T. M.: 1989, A task control architecture for mobile robots, in: *Working Notes of the AAAI Spring Symposium on Robot Navigation*.

Stone, P. and Veloso, M. M.: 1998, Towards collaborative and adversarial learning: A case study in robotic soccer, *Internat. J. Human–Comput. Systems* **48**.

Sutton, R.: 1990, Integrated architectures for learning, planning, and reacting based on approximating dynamic programming, in: *Proc. of the 7th Internat. Conf. on Machine Learning*, Austin, TX, pp. 216–224.

Tan, M.: 1993, Multi-agent reinforcement learning: Independent vs. cooperative agents, in: *Proc. of the 10th Internat. Conf. on Machine Learning*, Amherst, MA, pp. 330–337.

Veloso, M.: 1994, *Planning and Learning by Analogical Reasoning*, Springer, Berlin.

Veloso, M., Carbonell, J., Pérez, A., Borrajo, D., Fink, E., and Blythe, J.: 1995, Integrating planning and learning: The PRODIGY architecture, *J. Experim. Theoret. AI* **7**, 81–120.

Wang, X.: 1996, Planning while learning operators, in: B. Drabble (ed.), *Proc. of the 3rd Internat. Conf. on Artificia Intelligence Planning Systems (AIPS'96)*, Edinburgh, Scotland, pp. 229–236.

Watkins, C. J. C. H. and Dayan, P.: 1992, Technical note: Q-learning, *Machine Learning* **8**(3/4), 279–292.