Planning, Learning, and Executing in Autonomous Systems

Ramón García-Martínez
Departmento de Computación
Facultad de Ingeniería
Universidad de Buenos Aires
Bynon 1605. Adrogue (1846)
Buenos Aires. Argentina
email: rgm@mara.fi.uba.ar

Daniel Borrajo
Departamento de Informática
Escuela Politécnica Superior
Universidad Carlos III de Madrid
28911 Leganés (Madrid), España
dborrajo@ia.uc3m.es

Abstract. Systems that act autonomously in the environment have to be able to integrate three basic behaviors: planning, execution, and learning. Planning involves describing a set of actions that will allow the autonomous system to achieve high utility (a similar concept to goals in high-level classical planning) in an unknown world. Execution deals with the interaction with the environment by application of planned actions and observation of resulting perceptions. Learning is needed to predict the responses of the environment to the system actions, thus guiding the system to achieve its goals. 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 very few systems have focused on the acquisition of planning operator descriptions. In this paper, we present an integrated system that learns operator definitions, plans using those operators, and executes the plans for modifying the acquired operators. The results clearly show that the integrated planning, learning, and executing system outperforms the basic planner in a robot domain.

Keywords: Planning, unsupervised machine learning, autonomous intelligent systems, theory formation and revision.

1 Introduction

Autonomous intelligent behavior is an area with an emerging interest within Artificial Intelligence researchers [6, 10, 13, 14]. 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 [17]. 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 [5]. Thus, it is interesting to study how an autonomous system can automatically build planning operators that model its environment [6, 7]. In this context, machine learning applied to planning has mainly focused on learning control knowledge in many different ways such as: macro-operators, control knowledge, or cases. There is also currently a big trend on learning state transition probabilities in the context of reinforcement learning [15, 18]. However, very few have approached the generalized operators acquisition problem [3, 17], which is crucial when dealing with systems that must *autonomously* adapt to a changing environment.

We present in this paper a system, LOPE,[1] 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 [7].[2] In order to speed up the convergence, heuristic mutations of the observations have been used. Also, probability distribution estimators have been introduced to handle the contradictions among the generated planning operators. The learning technique integrates ideas from genetic algorithms (mutation as a learning operator), reinforcement learning (dealing with operator success probabilities and rewards), and inductive learning (generalization and specialization learning 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).

Section 2 describes the general architecture of the LOPE system, defining its architecture and top-level algorithm. Section 3 defines the representation that will be used in the paper for situations, observations and planning operators. Section 4 presents the learning model and its components. Section 5 defines the planner. Section 6 explain the performed experiments. And, finally, Section 8 draws the conclusions of the work.

2 General System Description

The integrated system learns, plans and executes in a simulated world according to the robot model described and used by many authors, such as [10, 12]. In this context, a model of the environment refers to a mapping between perceived situations, performed actions, and expected new situations, which is different from high-level models of the environment used by classical planners. The autonomous agent type of world are two-dimensional grids, where each position of the grids can have different elements, such as obstacles, energy points, or be empty. The objective of the LOPE system is to learn operators that predict the effect of actions in the environment, by observing the consequences of actions.

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

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

The system can be described as an exploring robot that perceives the environment, applies actions, and learns from its interaction with the world. At the beginning, the system perceives the initial situation, and selects a random action to execute in the environment. Then, it loops over a code that executes an action, perceives the resulting situation and utility of the situation (explained in section 4), learns from observing the effect of applying the action in the environment, and plans for further interactions with the environment. The top-level goal of the planning algorithm is implicit in the system: achieving a situation with the highest utility; 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 to the one that reflects other types of goals. Figure 1 shows an schematic view of the architecture, where the allowed actions is the set of actions that the robot can perform in the environment.

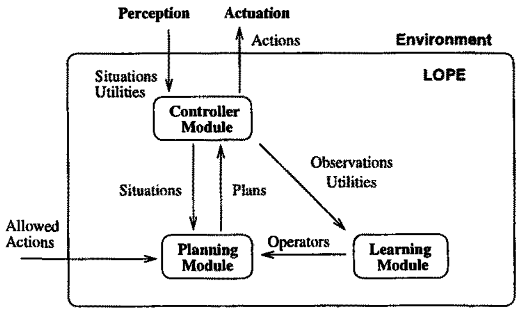


Fig. 1. Architecture of the Integrated System.

3 Representation

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 gets from its sensors. Its "states" are the inputs it receives from its sensoring system. Currently, there is no post-processing of its inputs for translating them into high-level descriptions. The model of the sensoring system is a modified version of the one proposed by [10], who suggested a system with 24 sectors, distributed in three levels. We use a model with eight sectors in two levels (close and distant sensing), and three regions (Left, Frontal, and Right) as shown in Figure 2. The values of each sector conform a binary vector of eight positions that describe each situation. A value of 1 in a position of the vector means that the corresponding sensor has detected something in its sector.

Previous work of the authors developed early versions of the learning mechanism [7, 8]. The representation was based on the model proposed in [6], in which an observation (also called experience unit) had the following structure:

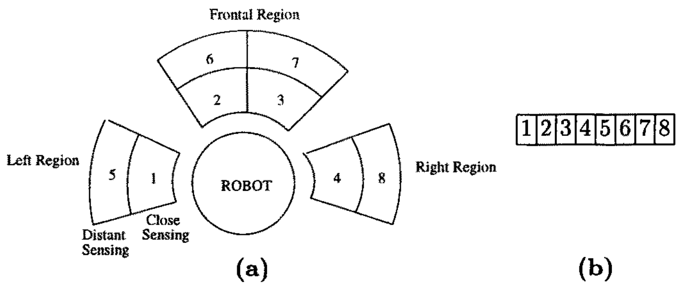


Fig. 2. Divisions of the sensoring system (a) and its internal representation (b)

[Initial Situation, Action, Final Situation]

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. The proposed planning operator model has the following structure and meaning:

PLANNING OPERATOR: O_i		
Feature	**Description**	**Values**
C	Initial Situation (conditions)	*s-vector*
A	Action	*action*
F	Final Situation	*s-vector*
P	Times that the operator O_i was successfully applied (the expected final 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 *s-vector* is a vector of eight positions and each position can have a 0, 1, or ?. The value ? means "it does not matter the value of that position". We have used the following actions for the value *action*: go, turn-left, turn-right, and stop. U is a function of the distance of the robot to the closest energy point. It is computed by the environment as

$$U(S,P) = \frac{1}{\mid 1 - d(S,P) \mid}$$

where S is the robot position, P is the closest energy point, and $d(S,P)$ is the distance between S and P. Then, this measure is given to the system as an input.

4 Learning Planning Operators

We will first define the concepts of similar and equal operators needed for the learning method, to further detail the learning method, present an example, and discuss the mutation 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 $o = [S_1, A, S_2]$, we say that:

- The two operators are **similar** if $C_1 = C_2$ and $A_1 = A_2$.
- The two operators are equal if $C_1 = C_2$, $A_1 = A_2$, and $F_1 = F_2$.
- The observation is **similar** to the operator O_1 if $S_1 \subseteq C_1$ and $A = A_1$.
- The observation **confirms** the operator O_1 if $S_1 \subseteq C_1$, $A = A_1$, and $S_2 \subseteq F_1$.

4.2 Learning Algorithm

Suppose a situation S_1 is perceived by the system, and there exists 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_2, the learning method processes this checking whether it is *similar* to any operator.

- If it is *similar*, it checks to see if the observation *confirms* the operator. Then, it rewards all such operators and punishes *similar* ones. If a *similar* operator exists, but there is none that is *confirmed* by the observation, it creates a new operator, punishes *similar* operators to the new one, and mutates those *similar* operators. The operators generated by the mutation procedure reward *equal* operators and punish *similar* ones.
- If it does not find a *similar* operator for the input observation, it creates a new one.

Punishing operators means incrementing the number of times that the pair (condition,action) of *similar* operators to O has been observed. The effect of incrementing their K is equivalent to punishing them. Also, *rewarding* operators means incrementing the P and K of a successful operator, with the equivalent effect of rewarding it. With respect to the utility, the system will record, for each operator, the utility of the highest-utility situation achieved by applying the operator action to the operator condition situation.[3]

4.3 Example of Learning Episodes

Suppose that the robot does not have any knowledge on how the environment will react to actions that are applied by the robot. We will see now how the system builds a set of operators from the observations o_1, o_2 and o_3, that do not have to necessarily be observed in consecutive instants of time, since other actions could have been applied in the middle.

$$o_1 = (00001001, \text{GO}, 00000000) \text{ with } U = U_1$$
$$o_2 = (00001001, \text{GO}, 00001111) \text{ with } U = U_2$$
$$o_3 = (00001001, \text{GO}, 00000000) \text{ with } U = U_3$$

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

After observing o_1, according to the algorithm described before, it generates a new operator O_1=[00001001,GO,00000000,1,1,U_1]. When it later observes o_2, it finds out that there is a similar operator, O_1. Thus, it first includes the new operator $O_2 = [00001001, GO, 00001111, 1, 2, U_2]$ into the set of operators $\mathcal{O}$. Then, it punishes the similar operators (only O_1 in this case), changing it to be $O_1 = [00001001, GO, 00000000, 1, 2, U_1]$. Then, it calls the mutation heuristics to create mutated observations. Among other mutation heuristics, the retraction heuristic, would generate the mutated observation $m = [00001001, GO, 0000????, U]$, where U would be U_1 in this case. Since it now finds that there are similar operators, O_1 and O_2, it adds a new (mutated) operator, $O_3 = [00001001, GO, 0000????, 1, 3, U_1]$, and punishes all similar operators by incrementing their K, leaving $\mathcal{O}$ as:

$$\begin{aligned}O_1 &= [00001001, GO, 00000000, 1, 3, U_1]\\ O_2 &= [00001001, GO, 00001111, 1, 3, U_2]\\ O_3 &= [00001001, GO, 0000????, 1, 3, U_1]\end{aligned}$$

For LOPE, the new mutated operator O_3 predicts what will be the final situation after applying the action GO to the initial situation 00001001 as well as O_1 and O_2 do. This is why $P_{O_3} = 1$. A value of three would mean a stronger relation between O_3 and O_1/O_2. When it observes o_3, which is a *confirmation* of operators O_1 and O_3, it rewards all operators equal to the observation (O_1 and O_3), and punishes all operators that are similar (O_2). Since it rewards both operators O_1 and O_2, the K of all operators gets increased to 5. The final set of operators is:

$$\begin{aligned}O_1 &= [00001001, GO, 00000000, 2, 5, U_1]\\ O_2 &= [00001001, GO, 00001111, 1, 5, U_2]\\ O_3 &= [00001001, GO, 0000????, 2, 5, U_1]\end{aligned}$$

4.4 Heuristic Mutation of Operators

The heuristic mutation of operators is based on the heuristics defined in [9] and [11]. Hayes-Roth proposed a set of heuristics for revising a faulty (buggy) theory, in the framework of theory revision. We selected which heuristics were applicable for the chosen vector representation, and transformed those for correcting violated expectations of plans.

- **Retraction:** generalizes an operator predicted situation so that it is consistent with the new observation.
- **Exclusion:** restricts the conditions of the operator, so that it does not apply to the observed situation again.
- **Avoidance:** also restricts the applicability conditions of the operator by conjoining negated preconditions that are sufficient for not predicting the observed situation again. Since the implementation of this heuristic in the chosen vector representation arrived to a similar procedure as the previous one, we did not use this heuristic.

- **Inclusion:** generalizes the operator conditions, so that it will later apply in the observed situation.
- **Assurance:** generalizes the conditions of the operator, so that it can be applied in the future to assure the predicted effects. Again, the implementation of this heuristic was similar to the previous one, so it was not used.

Salzberg heuristics are used to correct prediction violations. He proposed the following heuristics for revising predicting rules in a racing domain. We also transformed those heuristics to the vector representation. 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:** restricts the condition of an operator, so that it will not longer apply to the observed initial situation.
- **Ignorance:** assigns the fault of using an operator to unknown relations. Salzberg used a propositional representation with variable length, and since we are using a vector-based representation (fixed length representation), we could not use this heuristic.
- **Guilty:** has a set of predefined possible causes of fault in an operator, such that when a fault appears, the heuristic checks whether one of those causes is present in the observed situation. We did not implement this heuristic, since we could not find such a set of predefined causes, such as "when the first two bits are 1 in the situation", given that this domain does not have a rich domain theory.
- **Conservationism:** is a meta-heuristic that selects the mutation heuristic (from the Salzberg ones), that proposes less modifications in the conditions of an operator.
- **Simplicity:** is a generalization of the Hayes-Roth retraction heuristic in that it generalizes several operators into one.
- **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[4] that allows to efficiently reach the top-level goal of having the highest utility (being on top of an energy point) by being on a situation which yields such utility. In case another domain requires a more classical high-level set of goals (as in the case of the blocksworld

[4] In this case, the term operator and action are equivalent with respect to planning and execution, given that operators do not have variables.

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 the operator with the highest utility; that is, the operator that achieves the highest utility situation (its final 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. At the beginning, there are no operators, so the system generates a default plan by randomly selecting whether to act randomly, or to act by curiosity/exploration (by *approaching* a close obstacle).[5] If the system already has some operators, it builds a list of goals, each of them is a pair (situation,operator), where situation is the final 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 find a plan that can transform the current situation S into one of the subgoals. 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.

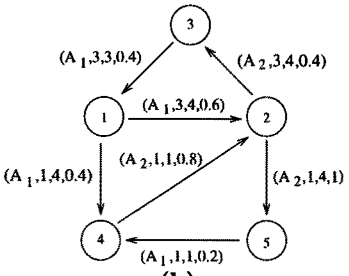
In order to create a plan to achieve a goal, the planner creates a graph by backward chaining on the goal. Since the goals are pairs (situation,operator), the root of the search tree will be the situation, that will only have one successor labeled with the operator of the goal. For each situation in 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.

As an example of how this algorithm proceeds, suppose that the system already built the set of operators shown in Figure 3(a), where there are two actions A_1 and A_2, and five situations S_1 to S_5. The search space corresponding to those operators is shown in Figure 3(b), 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 in subsection 5.2), and U is the operator utility.

[5] The system is close to an obstacle when there is a 1 on any element of the input vector.

$O_1=(S_1, A_1, S_2, 3, 4, 0.6)$
$O_2=(S_1, A_1, S_4, 1, 4, 0.4)$
$O_3=(S_2, A_2, S_3, 3, 4, 0.4)$
$O_4=(S_2, A_2, S_5, 1, 4, 1)$
$O_5=(S_3, A_1, S_1, 3, 3, 0.4)$
$O_6=(S_4, A_2, S_2, 1, 1, 0.8)$
$O_7=(S_5, A_1, S_4, 1, 1, 0.2)$

(a)



(b)

Fig. 3. Example of planning operators (a) and their graph representation (b).

Given that search space, if the planner tries to find a plan to achieve the higher utility reachable goal from the current situation S_1, it would first generate the list of pairs (goal-situation,operator) in descendent order of utility. In this case, the list would be:

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

As (S_5, O_4) has the highest utility level, the system builds the search tree shown in Figure 4, 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 in 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.

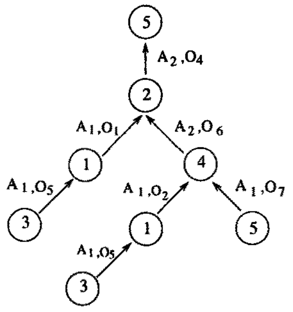


Fig. 4. Search tree generated when planning to achieve situation S_5 from S_1.

There are two plans that reach S_5 from S_1: $O_1 \circ O_4$[6] (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 $O_1 \circ O_4$ $(A_1 \circ A_2)$.

[6] $\circ$ represents the composition of operations.

5.2 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 ($S_j \subseteq C_{O_i}$) results in a situation S_k that verifies the predicted effects of the operator ($S_k \subseteq F_{O_i}$). We have shown in our previous work that this estimator is an unbiased estimator that follows a multinomial probability distribution. Therefore, the knowledge that the system has of the effects of an action A_i at a given instant can be represented by the transition matrix M_{A_i}, that has on the (j, k) position the quotient P_{O_i}/K_{O_i} of the operator whose action is A_i, its conditions are S_j, and the predicted effects S_k [2].

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 user-defined threshold. This property is critical when the system must act without supervision.

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:

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

From the analysis of M_{A_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 Experiments and Results

We performed two experiments to test the behavior of LOPE. In the first one, we averaged the results of running 50 experiments. In each experiment the initial setup (environment and position of the robot) was randomly selected, and LOPE performed 8000 cycles of learning, planning and execution. We compared four versions of the system: the base planner, in which operators are created directly from the observations, following Fritz *et al.* work [6]; the base problem solver using operators learned using heuristic mutation [7]; the base problem solver estimating for each operator its probability of success [8]; and the base problem solver, in which operators are mutated, and a probability estimator is assigned to each operator.

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 5 (a). These results clearly show that the combination of mutation and probability estimation outperform the base planner behavior, and, also, the separate use of any of them. The combined use of mutation and probabilities make the system converge towards a 80% of success plans, while the base planner converges towards half of it (around 40%).

The second experiment was performed to show the generality of the learned knowledge, as the knowledge transfer from one setup (environment and initial position of the robot) to another. We randomly generated a set of setups, $\mathcal{W}$, and averaged the results of running 50 times the following experiment: a setup w was randomly selected from $\mathcal{W}$; 8000 cycles were run on w; another setup $w' \neq w$ was chosen from the set $\mathcal{W}$; 8000 cycles were run on w', using the learned operators in w; and results were collected. The results are shown in Figure 5(b) where it is shown that the use of previously learned knowledge, even in another setup, improves the initial behavior as well as the convergence of the overall system, with respect to the results in Figure 5(a). For instance, while, in the first experiment, the 70% of success plans was achieved at around 3200 cycles, in the second experiment, the same success ratio was achieved at 1500 cycles.

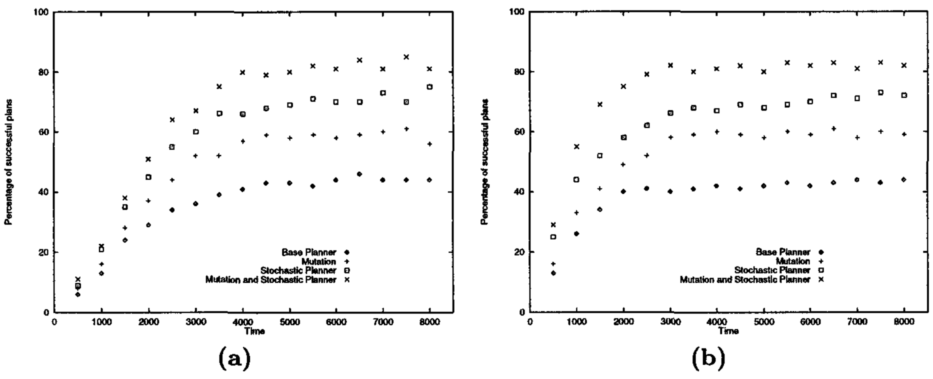


Fig. 5. Results of comparing four versions of the system with respect to successful planning (a) and knowledge transfer.

7 Related Work

The GINKO system [1] and the LIVE system [12] integrate perception, action and learning. They both differ from the proposed architecture in the fact that they do not take into account reinforcement nor heuristic-based refinement of operators. Christiansen [4] 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 refinement process. DYNA [15] integrates reinforcement learning, planning and reacting based on approximated dynamic programming. It differs from our work in the fact that the reinforcement procedure is local to an operator, while, in our case, the reinforcement of an operator explicitly implies the punishment of similar ones (global reinforcement).

OBSERVER [17] 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 [16] 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 system, with low-level planning.

8 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, defined 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 mutation 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.

References

1. M. Barbehenn and S. Hutchinson. An integrated architecture for learning and planning in robotic domains. *Sigart Bulletin*, 2(4):29–33, 1991.
2. Calistri-Yeh. *Classifying and Detecting Plan Based Misconceptions for Robust Plan Recognition*. PhD thesis, Department of Computer Science. Brown University, 1990.

3. Jaime G. Carbonell and Yolanda Gil. Learning by experimentation: The operator refinement method. In R. S. Michalski and Y. Kodratoff, editors, *Machine Learning: An Artificial Intelligence Approach, Volume III*, pages 191–213. Morgan Kaufmann, Palo Alto, CA, 1990.
4. Allan Christiansen. *Automatic Acquisition of Task Theories for Robotic Manipulation.* PhD thesis, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, March 1992.
5. B. Falkenhainer. A unified approach to explanation and theory formation. In J. Shrager and Langley P., editors, *Computational Models of Scientific Discovery and Theory Formation.* Morgan Kaufmann, 1990.
6. W. Fritz, R. García-Martínez, J. Blanqué, A. Rama, R. Adobbati, and M. Samo. The autonomous intelligent system. *Robotics and Autonomous Systems*, 5(2):109–125, 1989.
7. Ramón García-Martínez. Heuristic theory formation as a machine learning method. In *Proceedings of the VI International Symposium on Artificial Intelligence*, pages 294–298, México, 1993. LIMUSA.
8. Ramón García-Martínez and Daniel Borrajo. Unsupervised machine learning embedded in autonomous intelligent systems. In *Proceedings of the 14th IASTED International Conference on Applied Informatics*, pages 71–73, Innsbruck, Austria, 1996.
9. Frederick Hayes-Roth. Using proofs and refutations to learn from experience. In R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, editors, *Machine Learning, An Artificial Intelligence Approach*, pages 221–240. Tioga Press, Palo Alto, CA, 1983.
10. S. Mahavedan and J. Connell. Automatic programming of behavior-based robots using reinforcement learning. *Artificial Intelligence*, 55:311–365, 1992.
11. Stephen Salzberg. Heuristics for inductive learning. In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, pages 603–609, Los Angeles, CA, 1985.
12. W. Shen. Discovery as autonomous learning from enviroment. *Machine Learning*, 12:143–165, 1993.
13. Reid Simmons and Tom M. Mitchell. A task control architecture for mobile robots. In *Working Notes of the AAAI Spring Symposium on Robot Navigation*, 1989.
14. Peter Stone and Manuela M. Veloso. Towards collaborative and adversarial learning: A case study in robotic soccer. *To appear in International Journal of Human-Computer Systems (IJHCS)*, 1996.
15. Richard Sutton. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *Proceedings of the Seventh International Conference on Machine Learning*, pages 216–224, Austin, TX, 1990. Morgan Kaufmann.
16. Manuela Veloso, Jaime Carbonell, Alicia Pérez, Daniel Borrajo, Eugene Fink, and Jim Blythe. Integrating planning and learning: The PRODIGY architecture. *Journal of Experimental and Theoretical AI*, 7:81–120, 1995.
17. Xuemei Wang. Planning while learning operators. In B. Drabble, editor, *Proceedings of the Third International Conference on Artificial Intelligence Planning Systems (AIPS96)*, pages 229–236, Edinburgh, Scotland, May 1996.
18. C. J. C. H. Watkins and P. Dayan. Technical note: Q-learning. *Machine Learning*, 8(3/4):279–292, May 1992.