Learning to Solve Complex Tasks by Reinforcement: A New Algorithm

Mario Martin, Ulises Cortés
Dept. Llenguatges i Sistemes Informàtics
Universitat Politècnica de Catalunya
C Pau Gargallo 5, 08028, Barcelona (Catalunya), SPAIN
mmartin@lsi.upc.es, ia@lsi.upc.es

Abstract

In this paper, a new approach for learning to solve complex problems by reinforcement is proposed. In order to solve complex tasks the system is guided by a teacher who previously proposes intermediate general tasks to learn. The learnt behaviors to solve these tasks are added to the system's set of actions increasing its skills until it is able to easily solve the desired complex task. This approach uses a new reinforcement learning mechanism, robust to ambiguous information and able to learn general behaviors. These mechanisms are studied, described and finally tested with a set of experiments in a complex environment.

Keywords: Machine Learning, Reinforcement Learning, Robotics, Reactive systems.

1 Introduction

One important issue in reactive systems research is the learning ability to solve complex tasks. Several mechanisms has been proposed to make reactive systems learn, for instance [16] [5] (for a review see [10]). These mechanisms consist in obtaining, usually by means of a trialanderror process guided by a reinforcement signal, a mapping between the set of possible perceptions and the set of actions, that describes an adequate behavior.

Unfortunately, these methods are not well suited to learn how to solve complex tasks. A task whose solution is formed by a long chain of actions can be roughly considered as complex. It is hard to solve by trialanderror these tasks because the number of possible chains of actions to consider grows exponentially with the length of the chain solution.

We propose that this limitation could be overcome by previously learning to solve more simple but general tasks that can be used to easy the solving of the initial problem. This approach implies a *developmental* learning process in which the system steps over different stages. In each stage, characterized by the set of actions the system knows, it learns new behaviors that can be used as new complex actions to solve more complex tasks in a higher stage.

This article develops this proposal with a new reinforcement learning algorithm. This new algorithm is needed for the inadequacy of other known algorithms in learning general behaviors in our framework.

2 Learning to Solve Complex Tasks

In this section, the features for a good learning method to solve complex tasks are discussed and considered in order to build our learning mechanism.

There have been several approaches in reactive systems learning research, mainly based on the idea of reinforcement. Reinforcement consists in an internal or external signal that indicates, in a concrete situation, whether the executed behavior is appropriated. The learning comes from the finding of a correct mapping between the available set of actions and the set of perceivable stimulus, that maximizes the positive reinforcement.

When trying to develop such a mechanism for learning to solve *complex tasks*, we must understand the problems the system will face. The complexity of a task is mainly conditioned by the number of actions that composes the solution and by the number of perceptual states the system must deal with.

On one hand, the problem solution consists in a sequence of actions which will receive a reinforcement signal when the entire sequence has been executed (delayed reinforcement). Without any initial knowledge, the reactive system has only the trialanderror procedure for trying to solve any problem. It is known that it is very hard to *learn by trialanderror how to accomplish goals which imply a long sequence of actions*. Then, the complexity of a task comes from the adequacy of the actions the system can take to solve it: an adequate set of general actions lets solving the problem in *few steps*. The system is built with a set of primitive actions and there can be some tasks too difficult to be learnt from the beginning just with it. In our approach, the system solves this impasse learning to solve more general actions which will let it improve its skills. The behaviors learnt to solve these tasks can be useful for the solving of more complex tasks (reducing the number of actions that compose the solution chain) and then can be included as new available actions to the initial set. The intermediate tasks the system must learn in order to solve a complex task will be given opportunely to the system by a teacher.

On the other hand, it is known from the debate Ginsberg-Chapman [8] [4] that, for solving of complex tasks, it is necessary to *generalize situations*. In complex tasks, the number of different situations is too large to consider a response for each of them. Chapman [4] shows that a limited world perception by means of *deictic* sensors can be indicated for reactive systems solving complex tasks, producing a very profitable generalization process. This limited perception implies that the information of the environment the system has, is always partial, *incomplete and ambiguous*. Most learning mechanisms, for instance Q-learning [18], suppose the system having complete information about its environment.

When these methods have to deal with incomplete information or ambiguities, the problem of *perceptual aliasing* [19] appears, disabling the learning process. Then, an important feature that our learning process must show is the ability for learning with limited world perception in order to solve complex tasks. This implies that to solve a problem, the system must learn to search and perceive the required information[1].

Finally, reinforcement is given to solve a concrete problem, but an important feature the learning mechanism must show is the ability of learning *general behaviors* (not particular solutions) to solve problems. Problems with the same general goal and similar conditions in the environment ought to be solved with the same learnt behavior. This feature is not usually fulfilled in reinforcement learning mechanisms that obtain an optimal solution for a determined given final configuration in a concrete and no varying environment. For instance, finding the way to go out of *any* room with incomplete perception is a general task. To obtain the shortest path to go out of a particular room is not a general task.

All the observations we have made define the main features of an adequate learning mechanism to solve complex tasks and will be considered in the following section.

3 A new learning mechanism: the first step

In this section, we expose the learning mechanism developed according to the requirements obtained in the previous section. This mechanism must be able to learn general behaviors and to deal with incomplete information. In order to build a hierarchy of behaviors, it also must show a good performance in learning *reliable* behaviors from a crisp reinforcement signal.

The architecture developed by the learning process consists in an indexed table. The entries of the table are formed by the state of the sensors (situation) used in the problem solving. The output is a set of statistical information to compute the "probability" to obtain, for each executable action, a positive reinforcement. This statistical information is composed of the number of successes and failures occurred depending on the execution or not execution of each action in that situation.

The probability to get a positive reinforcement when executing an action a, given the actual situation s, is estimated as follows [2]:

$$P(s,a) = \frac{S\_A(s,a)}{S\_A(s,a) + F\_A(s,a)} - \frac{S\_NA(s,a)}{(S\_NA(s,a) + F\_NA(s,a)) * (k-1)}$$

[1]This kind of activity has been called *animate perception* [1].

[2]This measure has been compared with other ones and presents a more successful performance. Particularly, this measure uses information about the number of successes when the action has not been activated, given a estimation more similar to a correlation that a conditional probability.

where $S\_A(s,a)$ is the number of successes when the action a has been executed in situation s, $S\_NA(s,a)$ is the number of successes obtained when the action has not been executed, $F\_A(s,a)$ is the number of failures when the action has been executed, $F\_NA(s,a)$ is the number of failures when the action has not been executed, and k is the number of activable actions in the s situation.

This estimation is only significative if the entire sequence of actions to solve the problem is obtained from a constant policy. Obviously, learning implies to change the policy until the right one is obtained. If estimations change linearly, then the estimations carry equally influential information about different policies. Due to this problem, the number of successes and failures when an action has or has not been activated has to be weighted in order to decay the influence of old policies over the estimation (a kind of "forgetting mechanism"). The weighting is done as follows:

$$S\_A(s,a) = \sum_{t=0}^{n} \delta^{(n-t)} \;\; SAexp_t(s,a)$$

where δ, $0 < \delta < 1$ is the weight that decay the importance of past experiences, n is the number of experiences in which situation s has been found, and $SAexp_t(s,a)$ is a function that returns one if the action a was executed with positive reinforcement in the experience number t (0 the older, n the newest), and returns 0 otherwise. In the same way, the weighting is done for the other statistics, S_NA, F_A and F_NA.

With all this considerations, the learning algorithm is as follows:

1. Repeat until there's an error[3] or a success (problem solving phase).

    1.1 Choose and execute the action with the highest probability to obtain a positive reinforcement from the actual situation. In case there is more than one, choose one randomly.

2. For every situation s that the system has faced while constructing the sequence of actions, and every activable action a do (learning phase):

    2.1 Actualize the statistics $S\_A(s,a)$, $F\_A(s,a)$, $F\_NA(s,a)$ and $S\_NA(s,a)$ (by adding 1) depending on the obtained reinforcement and the activation of the actions.

    2.2 Weight all the statistics for situation s, multiplying them by δ

[3] An error occurs when the system executes a dangerous action or when certain number of actions have been executed without any success.

The learning cycle stops when the system does not learn anything in a predetermined number of new problems presented. To determine when the system has finished its learning, we use a table called *policy*. This table keeps the information of the best action to execute in every situation. If this table does not change during a large number of experiments, the system is not learning anything new and the learning process stops. In this case, the knowledge of *how* to solve the task is kept in the policy table which will be used to increase the system's set of actions and to achieve solving more complex problems.

3.1 Relation of the new algorithm to other works

The proposed algorithm presents several differences respect to other well known reinforcement learning mechanisms, specially with the TD family [16], [18], [2]. Most of these reinforcement algorithms find optimum solutions for a task, for instance, finding the shortest path to goal position in a concrete maze.

Unfortunately, these mechanism are not well suited to learn *general* behaviors with *incomplete* perceptual information. Under these constraints, the concept of optimization loses its sense. For example, suppose the situation depicted in figure 1. Blocks world: the objective is to find a block of a desired color. The sensory system reports that there is a block upon the one where its visual focus lies. It also knows that there are more blocks at the left of the same block, but it does not know where is the block it wants. There are two possible actions: move the visual focus up or left. Which is the optimal action? It depends on where the desired block lies. Suppose that the desired block is over the one where its visual focus lies. Then the best action is to move up the visual focus. Suppose that is at the left side of the block. Then the optimal action is to move left. Nor the perceptual information nor the past experiences cannot help the system to decide which is the optimal action, because if it is learning a general behavior the two situations can occur. Then, when learning general behaviors with ambiguous perceptual information, there cannot be optimal policies.

We face this concept in a different way. The system will learn a behavior under a set of constraints that include the reasonable cost to solve the problem. If the problem is not solved with a predefined number of steps, the trial fails. This ensures that the system will obtain a good behavior (low cost) although not the better because there no exists a better policy (lower cost).

Other works in reinforcement learning deal with the perceptual aliasing problem by extending the TD methods. Some try to disambiguate the state where the system currently is by a memory of past actions [12], others by building a model of the environment [13], [3] and others avoiding the ambiguous states [19]. These methods cannot help in the mentioned situation for the task of finding a block. Memory cannot help because we can face this situation with independence of the past actions taken. Models of the world cannot help because those models are for a concrete setting and the solving of a general task implies the solving for different settings. Finally, learning the task of finding a block

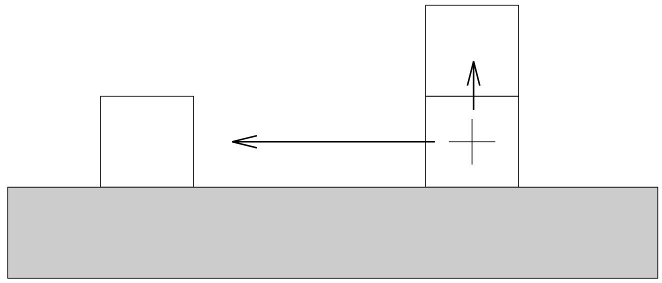


Figure 1: *There is no optimal action.*

must face with ambiguous states.

Another major difference between our method and others, as the TD family, is that the former does not need the process of *exploration*. Exploration is used to *optimize* the solutions learned by the system. Although a learned policy drives the system to the goal, it is necessary to test actions that there are not in the policy in order to find lower cost solutions. Exploration will only be useful in case of having a good policy but not totally reliable. In this case, a high rate of problems solved implies few penalties that are enough to change and improve the policy. To solve this problem without explore, a heuristic that forces to repeat the failed trials until learned is used (see section 5).

Another difference with TD methods is that they only actualize statistics of the activated actions for each situation, while our method actualize statistics for the all the actions, activated or not. With the success and failure information when the action is not executed (that we will call *negative* information) we can use a "probability" measure that is more similar to a correlation than to conditional probability.

As we have seen, the TD(λ) family is not well suited to the framework we are describing. The main problem of this approach is the perceptual aliasing problem that implies to face with a non-Markovian task [15]. This problem difficults to use as an estimation of the future reinforcement, the maximum estimation of the neighbor states, because the system does not know which are these states. Nevertheless, these methods are used in robotics in tasks with this perceptual problem.

An empirical comparison has been made between a close version of TD(0), Q-learning [18], and our method. It can be seen in section 4, figure 3. The Q-learning mechanism has been modified to make it resistant to perceptual aliasing and optimization. The modifications prevents exploration and annuls the need for optimization by setting $\gamma = 1$.

From the comparison we can extract the following conclusions. Q-learning shows a similar or even better results. This is due to the fact that Q-learning is an implementation of a TD(0) method (in which estimations are taken from neighbors) and our method extract the information directly from the reinforcement signal (ako TD(1) in this aspect). The first case has been showed slightly better for our task. Nevertheless, the problems arise when the system tries to learn a policy totally reliable. We found that there are problems specially difficult to learn, with a frequency bellow a threshold. These problems can never be solved because the punish associated for this low frequency and not solved problems, is not enough to change the police. To overcome this, we take the following rule: *when a problem is not solved, it is presented another time until it is solved.* Our algorithm works well with this heuristic, but Q-learning shows a degenerated performance (see experiences section).

We conclude that all this differences makes our algorithm more suited to learning complex tasks in *our framework* that pure TD methods, being able to deal with incomplete information and learn general tasks. The learning mechanism exposed will be used to learn general behaviors that in turn will be used as complex actions to solve more complex problems in a steeping process that is defined in the next section.

4 Building up: the second step

According to the exposed learning considerations, the possibility of learning to solve a complex tasks is subordinated to the set of actions available to the system. In order to achieve the needed skills for facing complex tasks, a teacher must guide the learning process giving a sequence of general tasks to be learnt to solve.

For every intermediate task the teacher must give the following information: the *objective* state and therefore when the positive reinforcement signal must be given; when an *error* has occurred (i.e., the maximum number of actions to be taken to solve a problem and the failure states); and the set of *useful* actions to solve a task from the whole set of possible actions, as well as the subset of sensors.

Obviously it is necessary to know when the system has accomplish the objective. It is also necessary to know when there has been no success. The failure is easy to characterize too. It appears when a number of actions have been executed and the objective has not been reached. The failure can also appears when a painful or dangerous action has been executed. Finally, the last information tries to ease the learning mechanism, indicating which is the set of useful actions and sensors to solve a task.

From this knowledge, a hierarchy of behaviors is built until the system can solve the complex initial task. An example of this construction is exposed in the next section.

The using of hierarchies of actions to solve complex problems is not a new idea. The approach we use is similar to the "shaping" of animats [6] or the construction of complex behaviors from low level ones [11] [14]. These methods has been used for learning from a immediate reinforcement or for learning optimal actions, but not for the learning of general behaviors from a delayed reinforcement. Other works as [7] [9] also learn to solve complex tasks by decomposing it not in simpler behaviors but in the state space. Nevertheless, in all these methods (including our method) is necessary to give a decomposition of the

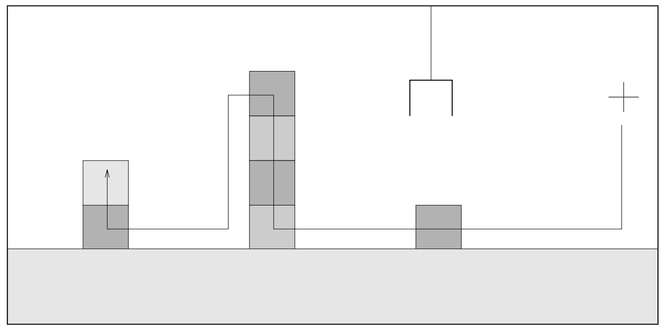


Figure 2: *A standard setting of the environment.*

task. We are planning to automatize this process with genetics algorithms to give us the necessary information for the construction of the hierarchy of behaviors.

The only work we are aware that automatically learns general behaviors from delayed reinforcement is [17]. Unfortunately, it is necessary to know how to solve complex tasks for find these general behaviors. Then, this method is useful for the *transfer* of learned behaviors but not for solving complex tasks.

5 Experiences

A set of experiments have been designed in order to show the performance of the proposed learning mechanism.

The environment selected is the "blocks world", that fulfills the required conditions of complexity: incrementally complex tasks can be proposed, the perceptual system gives incomplete or ambiguous reports, the initial set of actions is too simple to solve complex tasks, etc. This environment has also been used previously by Chapman [4] and Whitehead and Ballard [19] to show the possibilities and drawbacks of reactive systems in complex domains.

A standard setting of the environment (see figure 2) is composed of a window where we can see a table and blocks of different colors resting on it or forming stacks. In the picture we can also see the visual focus (in a standard learnt trajectory to find a block) and the hand of the system.

The system has been designed with a set of sensors that reports incomplete information of the world, and a very simple set of actions:

The set of sensors is composed of *visual sensors* as a `foveal vision`, a `peripheral vision`, and the `visual memory`; *proprioceptive*[4] *sensors* as the `Hand-Fovea angle` and the `Hand-Fovea distance`; and *tactile sensors* as the `content`, the `touching-something?`

[4]Proprioception is the direct perception of the relative position of parts of the body via, for instance, muscle tension.

and the `Open-or-closed?` hand.

Initially, the set of actions of the system is composed of: *actions over sensors* as `to move the visual focus one position`, `to move the visual focus to a mark` (visual memory), and `to put a mark`; and *Actions over effectors* as `to move the hand`, and `to open and close the hand`.

A kind of complex tasks that the system must solve consists in putting an unlocalized block of a given color in the position where the mark lies[5]. Initially, the system only has the mentioned set of actions and sensors. Then, a solution to the problem proposed will be composed of a large chain of actions that, as we have seen in section 2, is the main problem that prevents learning from a trialanderror procedure.

In order to solve the problem with our method, we will suppose that the teacher provides a set of general subtasks to learn which allows to solve the initial problem. In our case, the teacher proposes to learn the following general tasks: `Searching for a block`, `Moving the hand to the visual focus`, `Removing the top of a stack` and `Grasping a block`.

The learning process for each of these tasks generates a behavior that can be considered as a new general action. These new actions increase the repertory of actions of the system and can be used for solving more complex problems. In the following subsections we will show the experimental results of applying the learning mechanism proposed in section 3 for each of these tasks.

5.1 Task 1: Searching for a block

This task consists in searching for a block of a desired color from the set of blocks in the world. This is a non-trivial task, due to the limited perceptual information given to the system. To solve this task, the system must take advantage of the invariant features it finds experimenting in the environment, as the empirical fact that blocks rest on the table or over other blocks.

In order to solve this task, there is not necessary the whole set of sensors and actions. The teacher may suggest the foveal vision and peripheral vision as the useful sensors, and to move the visual focus up, down, left and right as the useful actions. The system will receive a positive reinforcement when it perceives by its foveal vision a block of the desired color, but it will receive a negative reinforcement when it does not reach the goal in 25 actions or when the visual focus goes out of the environment window.

The results obtained for our algorithm and for Q-learning in the `searching for a block` task can be seen in figure 3. It represents the number of successes obtained (y axis) for each hundred of settings presented to the system (x axis).

The learning cycle consists in presenting randomly generated new situations (settings) of the environment to the system from which to solve the task.

[5] With this action, substituting colors by letters, it is not very hard to solve the fruitcake problem [4].

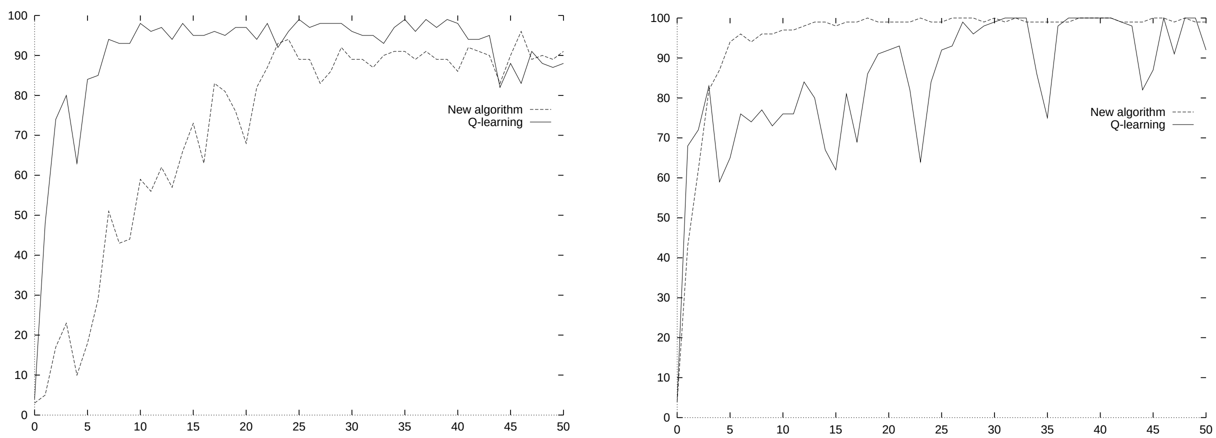


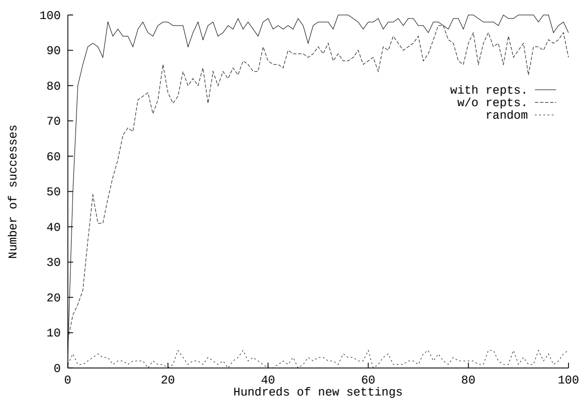
Figure 3: *Q-learning vs the new method: a) w/o repetitions. b)with repetitions.*

In the graphic for this task (figure 3a), the learning of the system can be observed. Near the 2.000 presentations of different settings the learning rate is stabilized at the 90% of successes. This percentage shows that there are a 10% of very complex problems the system is not able to solve. In this case, the system has found a good policy and it considers that it is not useful to change this policy for solving this little percentage. A random behavior can only reach a 4% of succeses.

In a second experiment (figure 3b) the system is forced to repeat the settings it has not solved, until it has a success or the number of repetitions exceeds a predetermined number. The repetition of not solved experiments punishes the policy function and forces the system to find a solution for them. The results obtained, can also be seen in the graphic. There are counted as successes only the problems solved at the first trial. A repetition of a setting it is not considered as a new setting. As it can be noted, the solving rate shows a fast growing for our method that reaches the 95% of successes with 500 different settings and finally achieves a stabilized 100% of successes with 50.000 settings. This experiment shows that our method learns a reliable behavior, while the Q-learning method shows a degenerate performance.

5.2 Task 2: Moving the hand to the visual focus

This task consists in learning a general action to coordinate the hand and the eye of the system. This action must be able to move the hand towards the place where the visual focus lies. The hand, as depicted in figure 2, is hanging on the world ceiling and thus, it can only reach the visual focus if it is situated on the vacuum or over a cleared block. While moving, the hand must avoid colliding with any stack. The teacher suggests the fovea, touching-something?, hand-fovea distance and hand-fovea angle as the useful sensors, and to move the hand as the useful actions to solve this task.



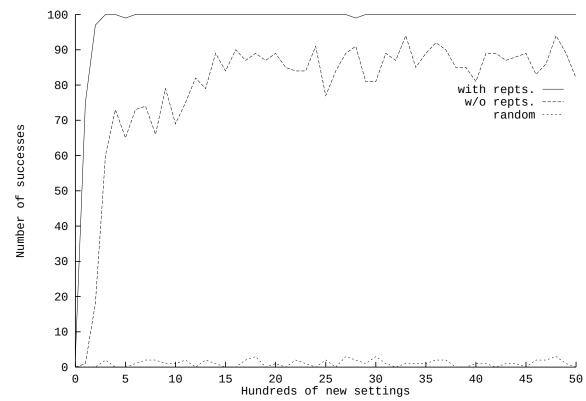


Figure 4: *a) Moving the hand to the visual focus, b) Removing a block*

A first experiment for this task was a presentation of settings to the system with the learning ability inhibited. In this case, the system could only solve the problem choosing randomly an action for each situation (random curve). As we can see in figure 4, the system learns to solve the proposed task. The performance without learning shows a success rate of 2%. The learning without repetition of failed settings reaches the 87% of successes. Finally, the learning with repetition of failed settings achieves with 700 settings the 90% of successes.

5.3 Task 3: Removing the top of a stack

This task consist in removing a block from the top of a stack. At the starting state, the fovea and the hand are placed in the target block. The block must be removed by the hand, avoiding to collide with any other block.

The teacher suggests the nearness and open hand as the useful sensors, and to move the hand, to close hand and to open hand as the useful actions to solve this task.

This task is easily learned by the system, as can be seen in figure 4. With only 300 settings, the number of successes reaches the 95%.

5.4 Task 4: Grasping a block.

In this task, the system must take a block with the hand. At the initial situation, the visual focus is on the target block. This is a non-trivial task because the target block can be cleared or placed on the middle of a stack. In the last case, the system must learn to unstack the blocks placed over the target block.

To learn how to solve this task, the fovea, peripheral vision, open hand and content sensors, and the actions to move the visual focus, to move the hand towards the visual focus and to remove a block are selected by the teacher.

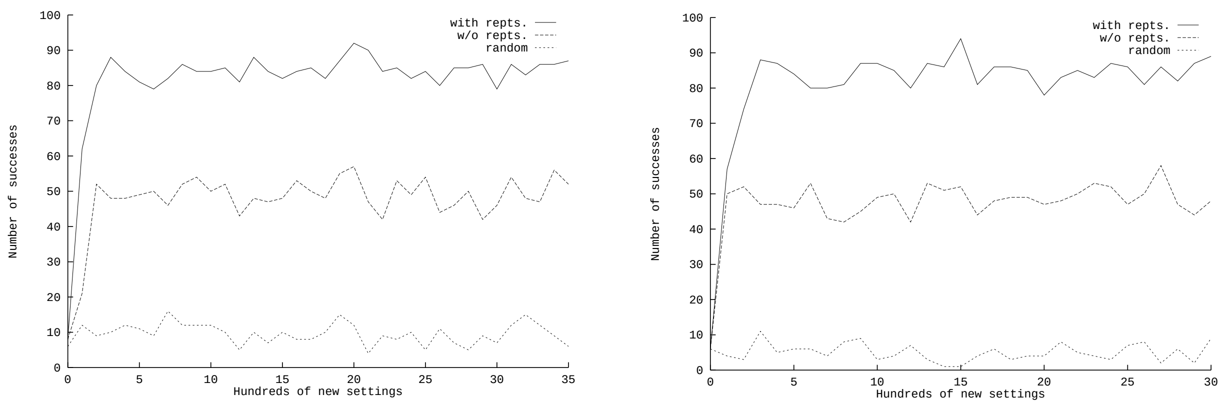


Figure 5: *a) Grasping a block, b) Final task*

This is the first behavior that uses previously learned actions to solve the task. This can be noted in figure 5 which shows the learning performance for this task. The results for learning with repetition of failed settings can only achieve the 85% of successes. This is mainly due to the use of actions previously learned that does not have a 100% of successes.

5.5 Final task: Moving a block to the mark.

This is the initial task proposed to the system. The task consists in moving a block of a desired color to the place where the mark lies. To solve this task, the system must find the block, grasp it and move it to the target place.

In this case, the teacher selects to move the visual focus to the mark, to move the hand towards the visual focus, to grasp a block and to close the hand as the useful actions, and the fovea, open hand and content as the useful sensors to solve the task.

Figure 5 shows the results of the learning process. The solving rate is stabilized at about 85% of successes.

6 Conclusions

In this paper, a mechanism for learning to solve complex tasks by reinforcement has been developed and tested. This research lead us to several conclusions:

On one hand, we have shown that it is possible to learn how to solve complex tasks. The final task proposed in the "blocks world", considering the initial set of actions the system knows, is not currently resolvable by any other direct learning method. The success of our proposal is due to the use of a learning mechanism robust to ambiguous information, that can improve the abilities of the system by learning new behaviors useful to solve general tasks.

Nevertheless we must note that several aspects can be improved. One of them is the limits observed for learning in tasks 4 and 5, that are caused by the use of learnt actions not completely experienced. This problem could be solved enhancing the tuning, of the previously learnt behaviors, when the task where they are used, has reached a high percentage of successes. Another one is that the guide the teacher must provide is excessively fitted to the intended task to learn. We are currently trying to solve this problem developing an automatic mechanism, based on genetic algorithms, that decide the information the system will be given to solve a complex task.

References

1. D. Ballard. Reference frames for animate vision. In *Proceedings of IJCAI89*, 1989.

2. AG Barto, RS Sutton, and CJ. Watkins. Learning and sequential decision making. Technical Report TR-89-95, University of Massachusets, 1989.

3. A.R. Cassandra, L.P. Kaelbling, and M.L. Littman. Acting optimally in partially observable stochastic domains. In *Proceedings of Twelfth National Conference on Artificial Intelligence*, 1994.

4. D. Chapman. Penguins can make cake. *AI Magazine*, 10:45–50, winter 1989.

5. D. Chapman and L. Kaelbling. Learning from delayed reinforcement in a complex domain. Technical Report 90-11, Teleos Research, 1990.

6. M. Colombetti and M. Dorigo. Robot shaping: Developing situated agents through learning. Technical Report 92-040 revised, International Computer Science Institute, 1993.

7. P. Dayan and G.E. Hinton. Feudal reinforcement learning. In *Proceedings of NIPS'5*. Morgan Kaufmann, 1993.

8. M.L. Ginsberg. Universal planning: An (almost) universally bad idea. *AI Magazine*, 10:41–44, winter 1989.

9. L.P. Kaelbling. Hierarchical learning in stochastic domains: Preliminary results. In *Proceedings of the Tenth Machine Learning Workshop*. Morgan Kaufmann, 1993.

10. L.P. Kaelbling. *Learning in Embedded Systems*. MIT Press, 1993.

11. L.J. Lin. *Reinforcement Learning for Robots Using Neural Networks*. PhD thesis, Carnegie Mellon University, 1993.

12. L.J. Lin and T.M. Mitchell. Memory approaches to reinforcement learning in non-markovian domains. Technical Report CS-92-138, Carnegie Mellon University, 1992.

13. R.A. McCallum. Learning with incomplete selective perception. Technical Report TR-453, University of Rochester, 1993.

14. P.S. Singh. Transfer of learning by composing solutions for elemental sequential tasks. *Machine Learning*, (8):323–340, 1992.

15. M.I. Jordan S.P. Singh, T. Jaakkola. Learning without state-estimation in partially observable markovian decision processes. In *Proceedings of the Eleventh Machine Learning Workshop*. Morgan Kaufmann, 1994.

16. R.S. Sutton. Learning to predict by the methods of temporal differences. *Machine Learning*, 3(1):9–44, 1988.

17. S. Thrun. Finding structure in reinforcement learning. In *Proceedings of NIPS'7*, 1995.

18. C. Watkins. *Learning from delayed rewards*. PhD thesis, Cambridge Univ., 1989.

19. S.D. Whitehead and D.H. Ballard. Learning to percive and act by trial and error. *Machine Learning*, 7:45–83, 1991.