IJCNN '99 Tutorial

1999 International Joint Conference on Neural Networks

Washington, DC - July 10-16, 1999

Track 5: Robot Control and Neurocontrol – course 5A

Neural Networks and Q-Learning for Robotics

Claude F. TOUZET

Center for Engineering Science Advanced Research

Computer Science & Mathematics Division

Oak Ridge National Laboratory

TN, USA

touzetc@mars.epm.ornl.gov

Introduction

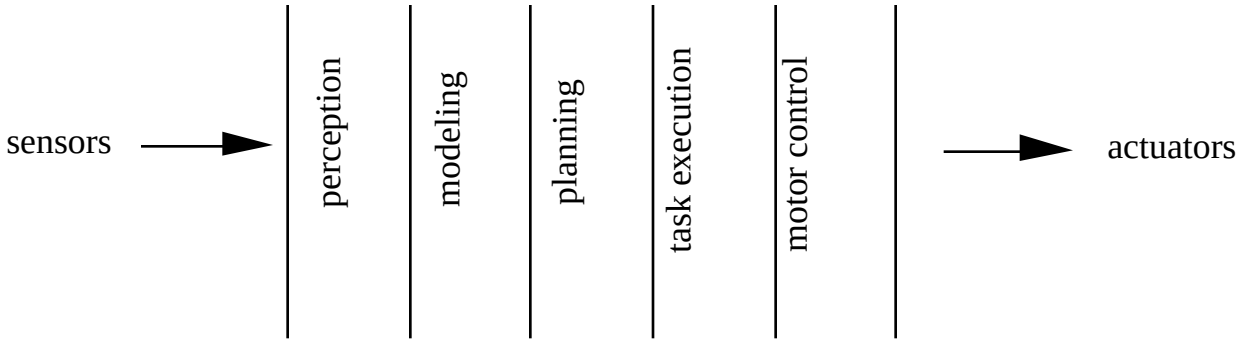


Fig. 1. Classical decomposition of an autonomous robot.

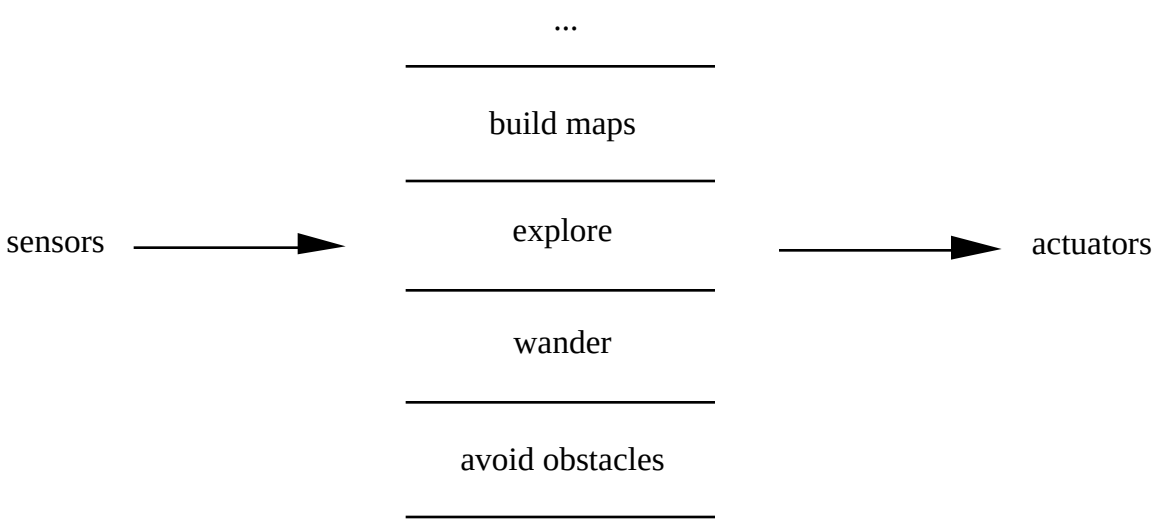


Fig. 2. Behavior-based decomposition of an autonomous robot

Supervised learning of a behavior (fig 3), such as implemented by the gradient back-propagation algorithm (fig. 4), requires the definition of a representative set of learning examples of such behavior. Each example is a couple (input, desired output) and the goal of the algorithm is to reduce the quadratic error. This error is the difference between the desired output value and the obtained value. The back-propagation learning algorithm contributed in a major way to the popularity and the diffusion of connectionist applications and the multi-layer perceptron model. However, it is important to note that the model of the desired behavior is in some sense hidden in the learning base: the learning base has to be meaningful if we want the synthesized neural behavior to be effective.

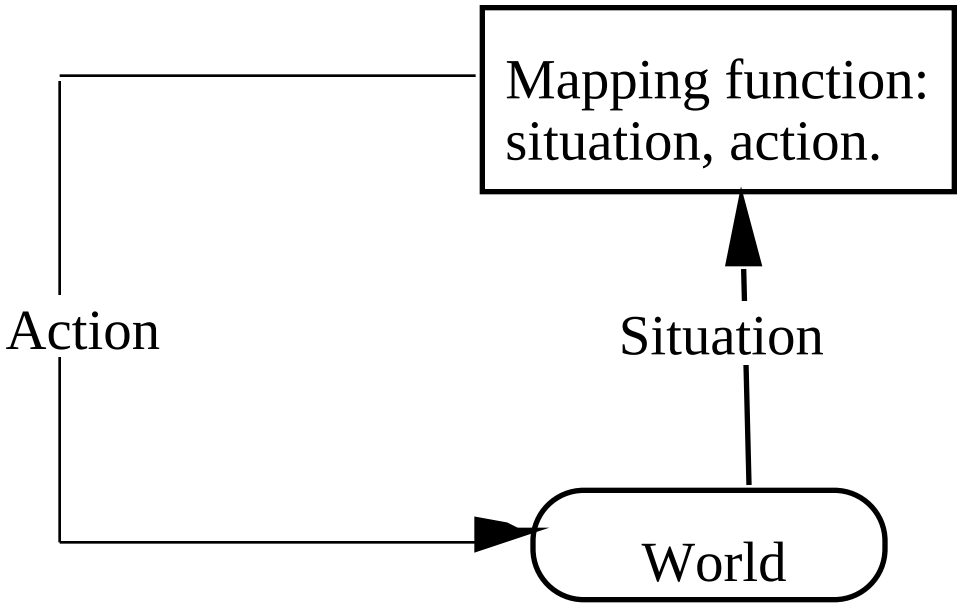


Fig. 3. A behavior: a mapping function between situations of the world as sensed by the sensors and actions undertaken by the actuators.

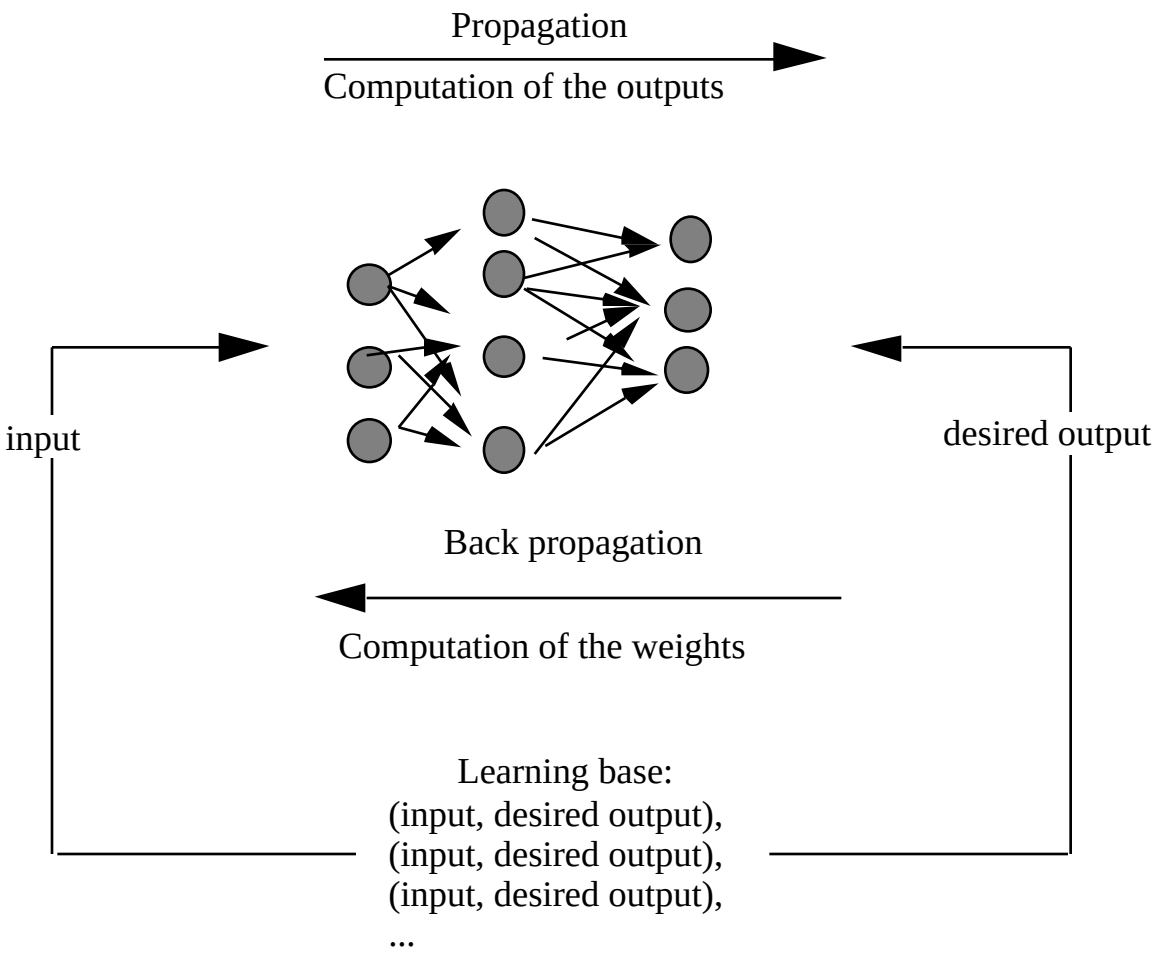


Fig. 4. The backpropagation algorithm modifies the network weights so as to reduce the error between the output of the network and the desired output. To this end, the desired output must be known, so as to be able to compute a quantitative error. The learning set is composed of input-desired output pairs . The learning is iterative, multiple presentations of the learning base are made to the network (from a few hundreds presentations to hundreds of thousands). The network weights are modified at each presentation, unless the output error is null.

Automatic generation of the learning base: [Heemskerk 96] proposes to equip the robot with a built-in avoidance behavior when the behavior to learn is obstacle avoidance. In addition to the built-in control structure, the experimenter intervenes when it is clear that the robot is about to collide. In these cases, the learning data are collected by hand-driving the robot on a collision free course. When the robot does collide, the input-output pairs leading to the collision are removed from the learning base (fig. 5). The learning examples are used to train the neural network off-line before testing the robot in the real world. As reported, "many different learning bases were collected and several neural networks were trained in which the number of hidden units and the learning parameters were varied. Successful performance was obtained for some of these networks."

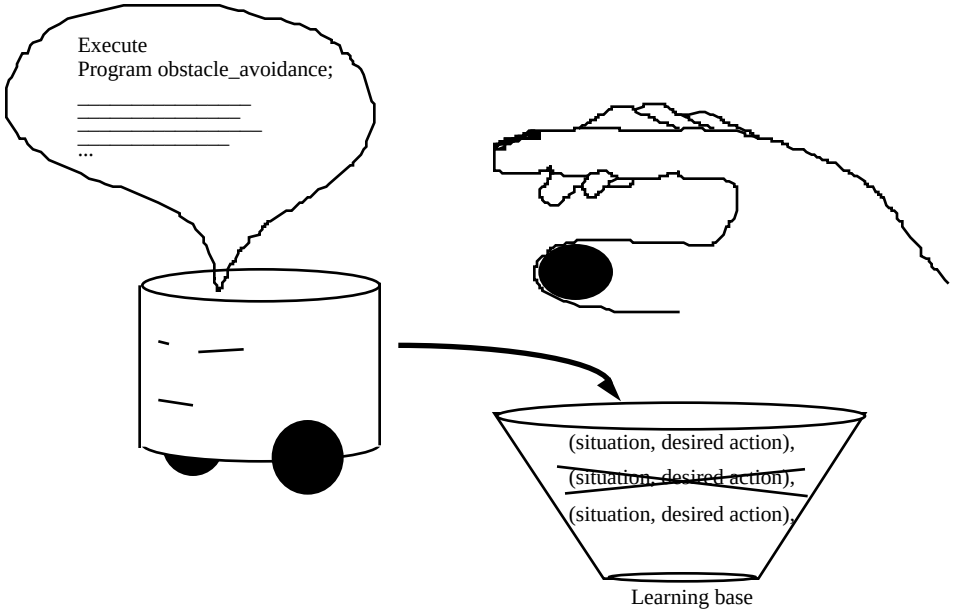


Fig. 5. Generating the learning base: a programmed avoidance behavior generates the learning base examples, which are eventually removed or completed by the experimenter.

- The training by reinforcement is a possible alternative to the definition by the operator of the learning base. The primary difference with supervised learning lies in the form of the examples of training. They are triplets (situation, action, utility), where the last component encodes the utility to perform this "action" in this "situation". The learning examples are generated automatically during a phase known as "exploration". It is generally a random search in the state space, the large size of space does not allow a complete cover (exhaustive) and makes it necessary to invoke generalization techniques (in our case, artificial neural networks). Also, the “utility” must be computed automatically for every visited (situation, action) pairs. This can be done using an equation, a rule of calculation or a procedure. It is at this level that the intervention of the operator is needed. The success of the application will depend on the quality of the function specifying the utility of a pair. This function, usually called reinforcement function, measures the performance of the system. The performance is defined as the utility of the proposed actions relatively to the task to be achieved. The utility is a qualitative concept, generally coded in a binary way: +1 = good, -1 = bad and 0 when one is able to decide. Usually, the generation of the learning base is done in parallel of the exploitation, therefore the training is incremental. This is why, when a representative learning base is finally built, the learning is finished.

To exchange the "manual" building of the learning base against a simple measure of the performance is certainly economic for the operator, but the price to be paid is a slower convergence of the learning. Indeed, the utility of a particular action in a particular situation is a less rich information than that of knowing exactly which action has the reatest utility (i.e., the desired action). The goal of reinforcement learning is thus to find the output of greatest utility by using the "binary" utilities associated to each available examples. In complement to reinforcement learning, there is always a learning algorithm specific of the neural network used as the implementation tool (e.g., back-propagation for multilayer perceptrons, Kohonen algorithm for a self-organizing map).

It is important to note that supervised learning and reinforcement learning are not competing with each other. Each one corresponds to a particular niche of applications, according to whether one has a representative set of examples, or just a measure of the performance of the required behavior.

In 1999, one of principal application domains for reinforcement learning is autonomous robotics, a domain not addressed with supervised learning because of impossibility to model the real world (i.e., to know the desired action) with enough precision so as to be able to account for the heterogeneity of the sensors (fig. 6 & 7), the ambient noise and the dynamics of the robot-world relation.

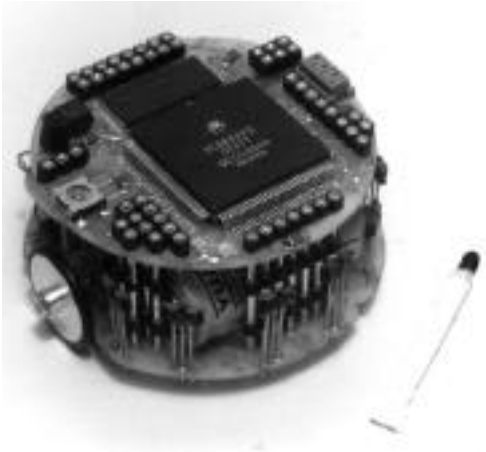


Fig. 6. The miniature mobile robot Khepera.

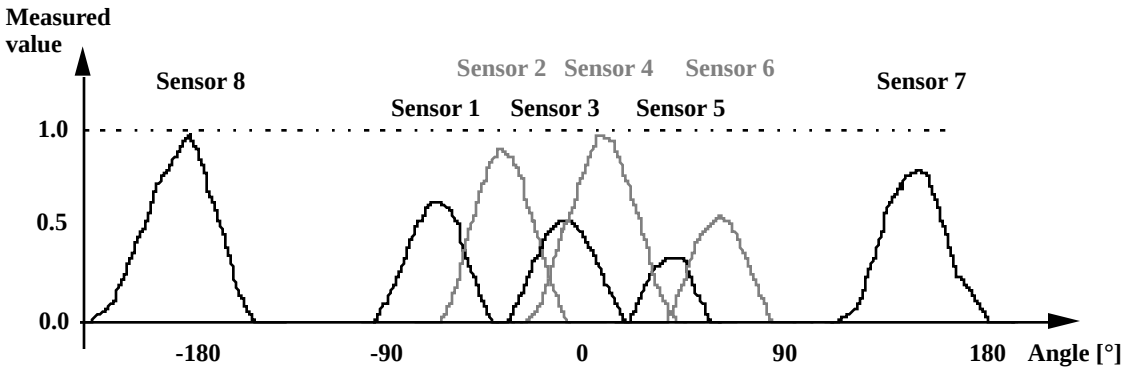


Fig. 7. Sensor values with an obstacle moving around the robot at a distance of 2 cm. The angle on the X axis is measured between the forward direction of the robot and the direction of the obstacle.

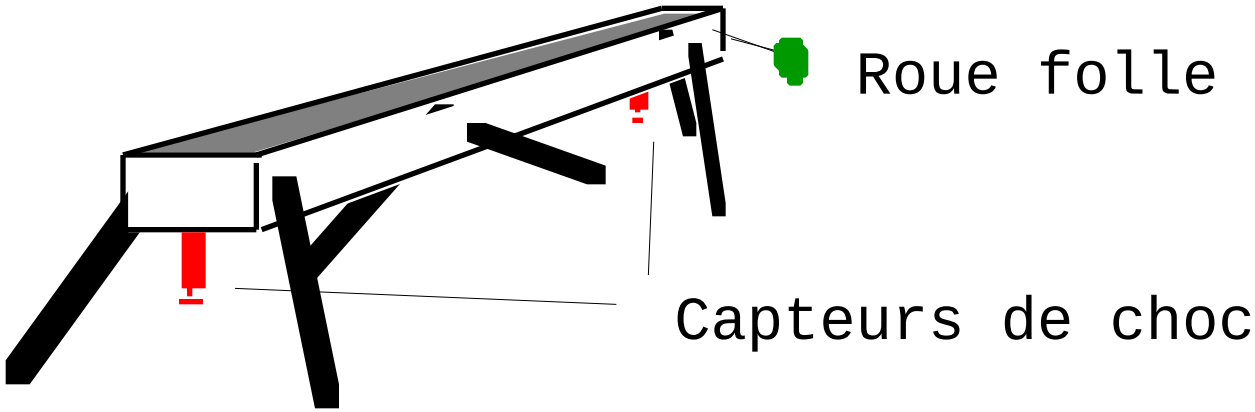


Fig. 8. Hexapod robot with sensors for the RF in a learn-how-to-walk experiment.

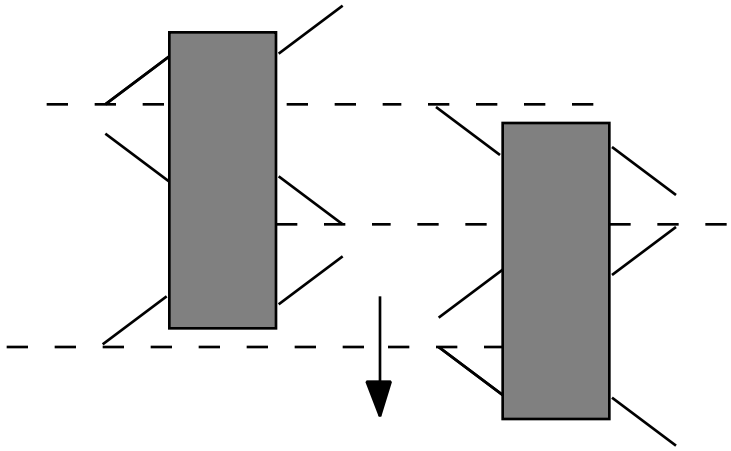


Fig. 9. A two-step walk.

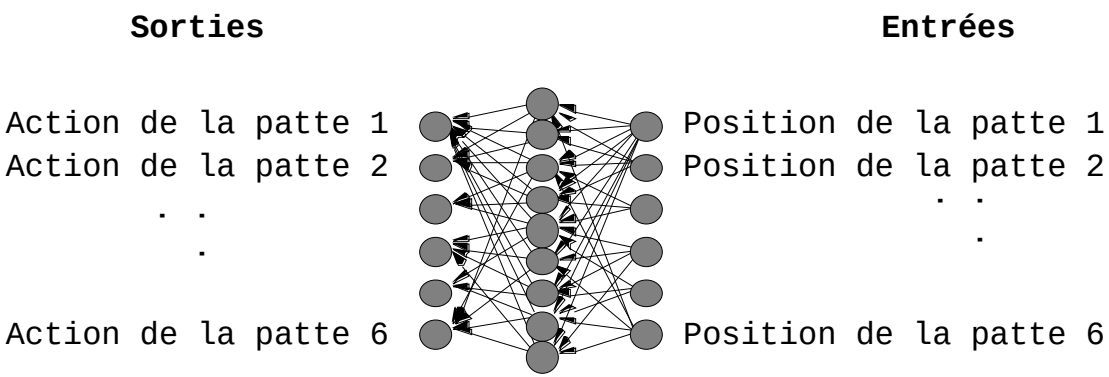


Fig. 10. Neural Network Architecture.

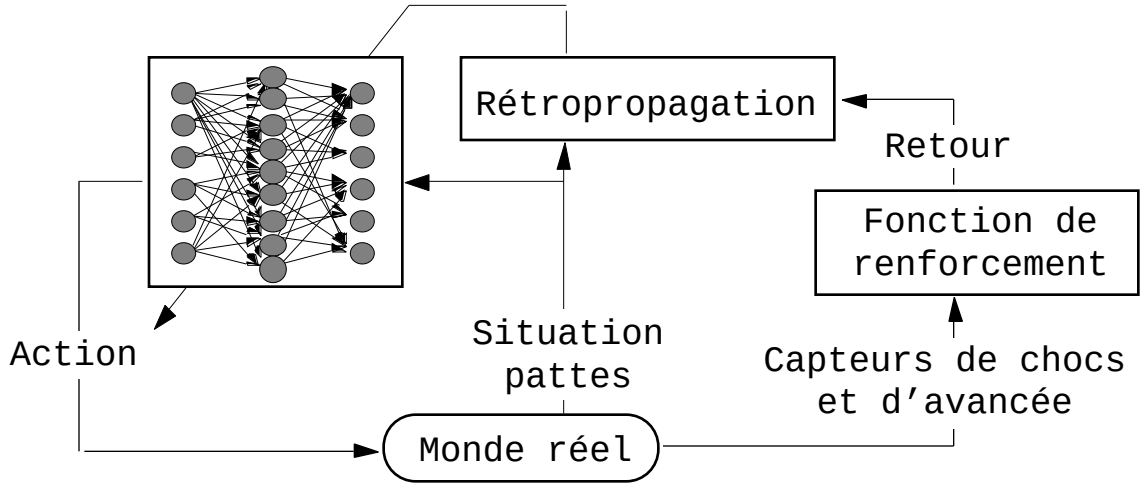


Fig. 11. Reward-Penalty Learning Algorithm

Questions:

- Exploration, how to choose the actions? How much randomness?
- Exploitation, when should we stop exploration ? How to generalize?
- RF design, what sensors?
- How to deal with delayed information?

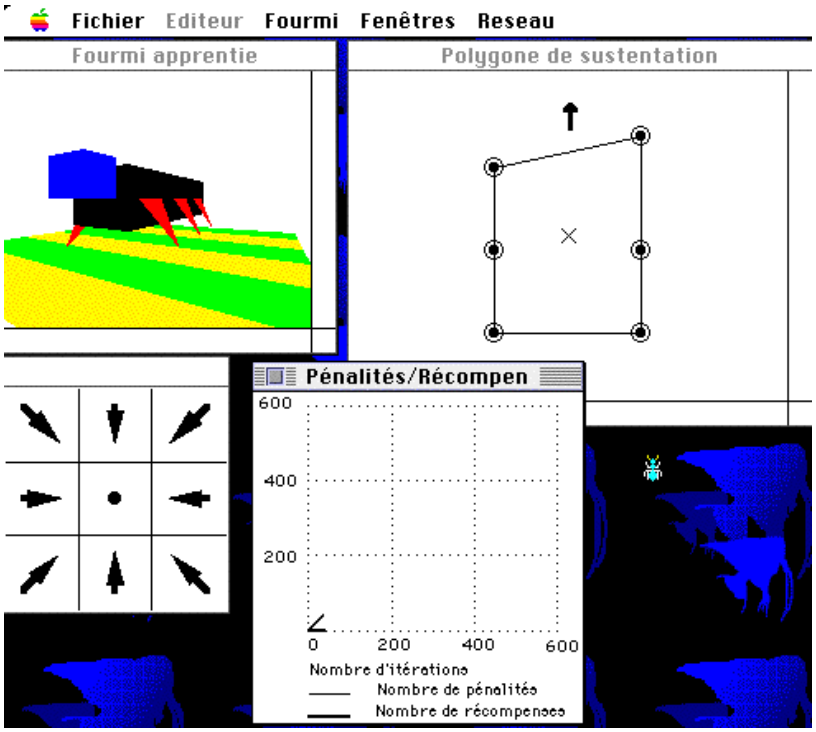


Fig. 12. ANTROID, winner French National Apple Competition, 1993.

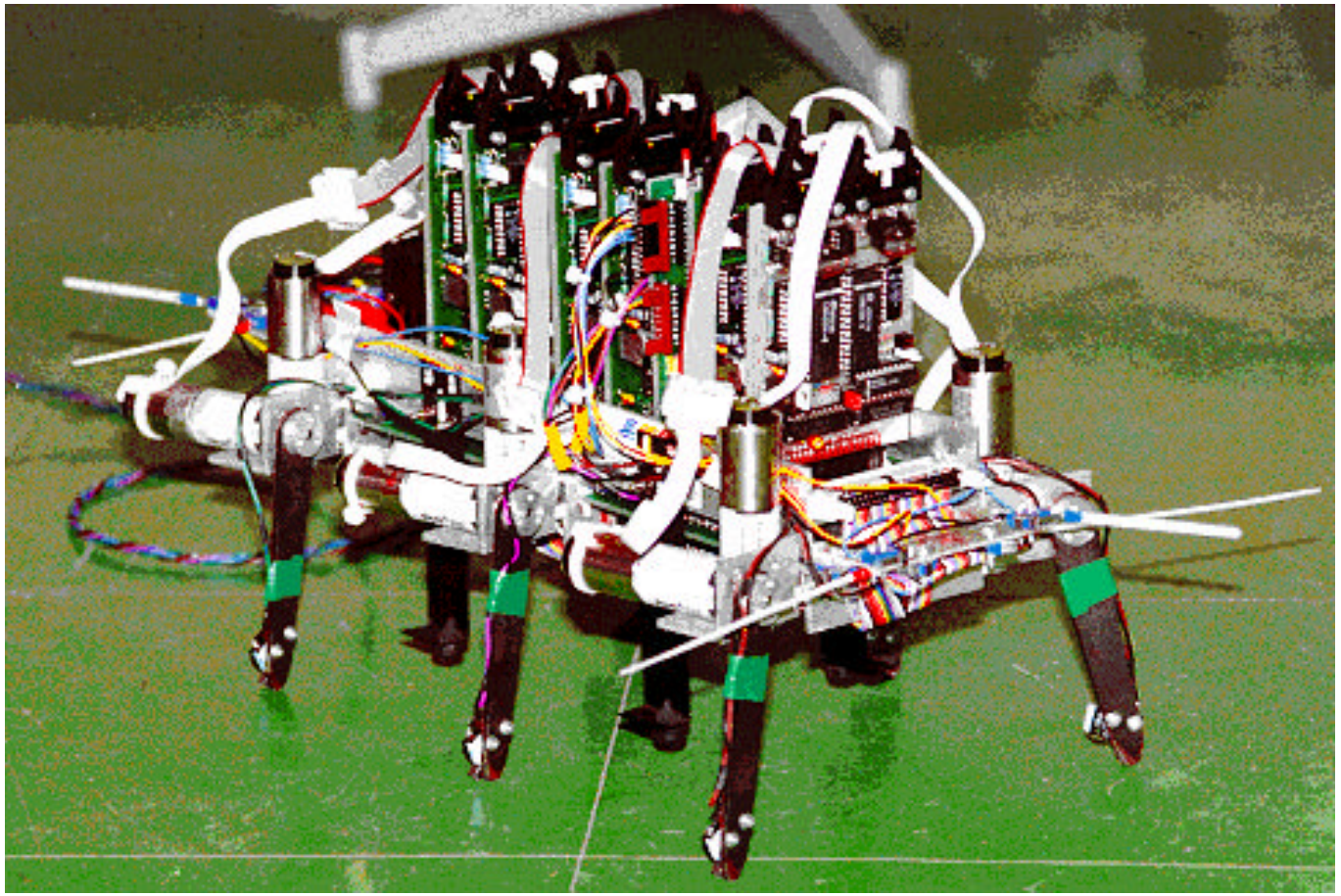
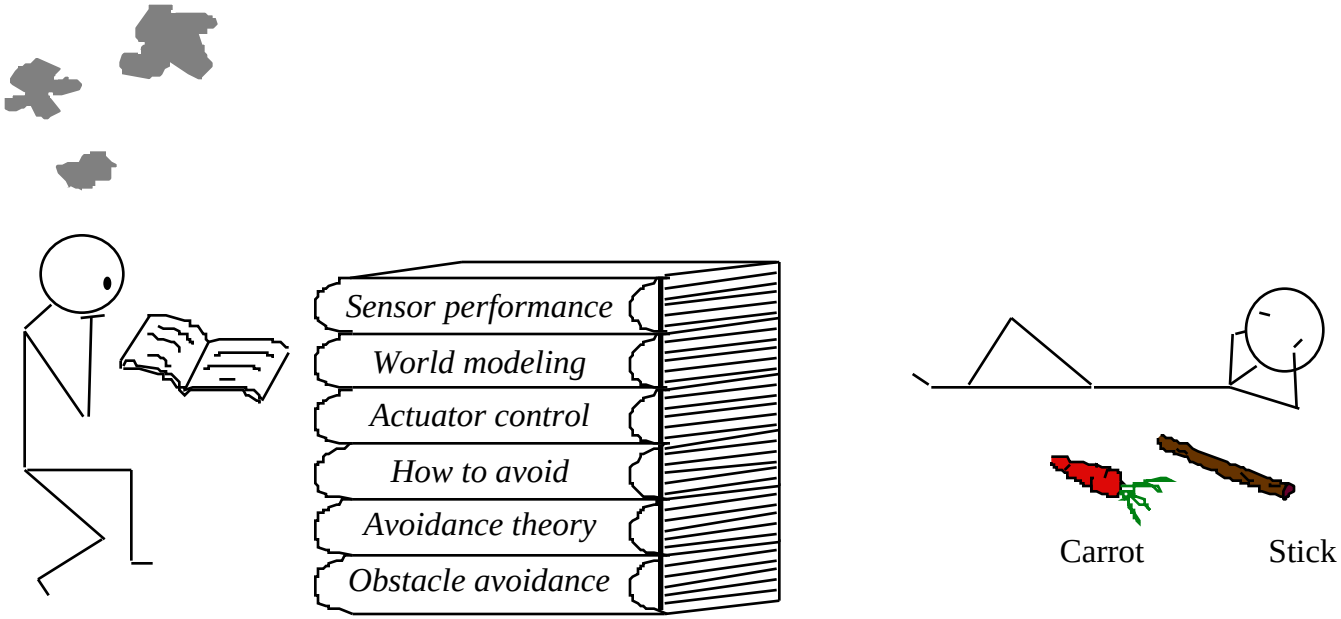


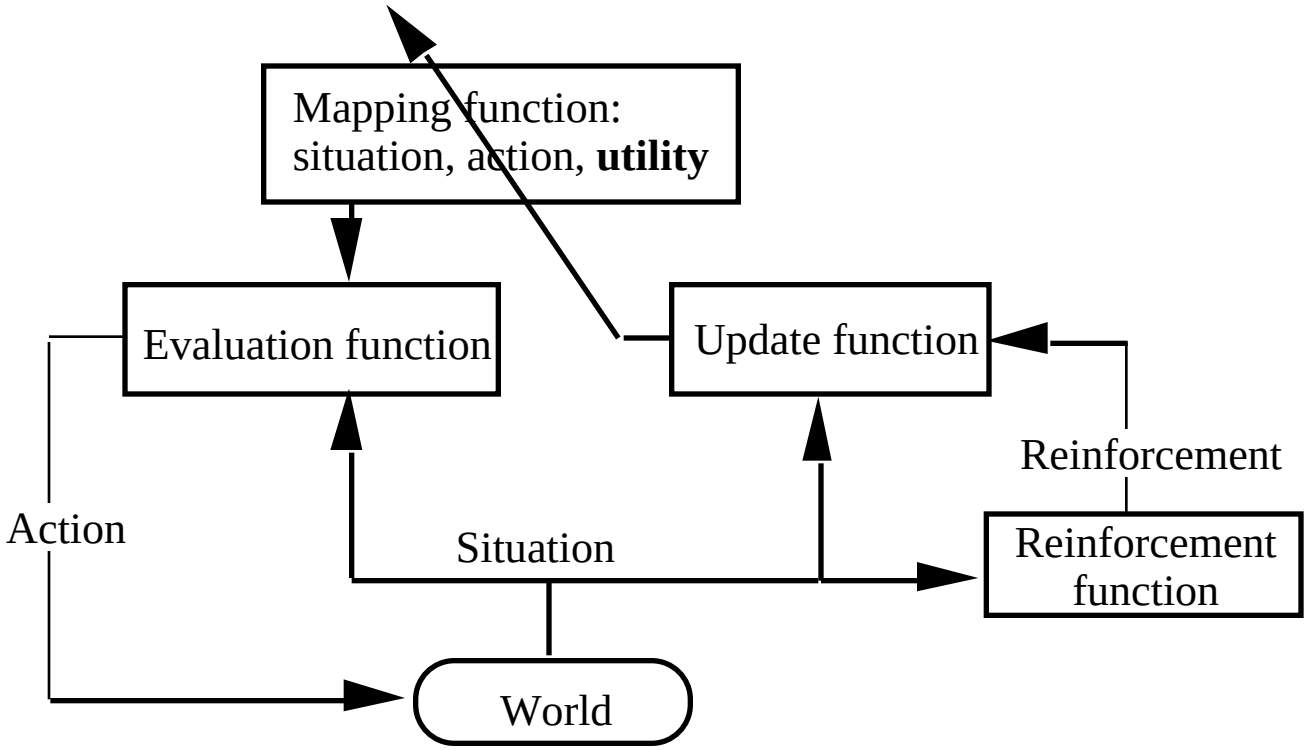
Fig. 13. Actual robot, 1994.

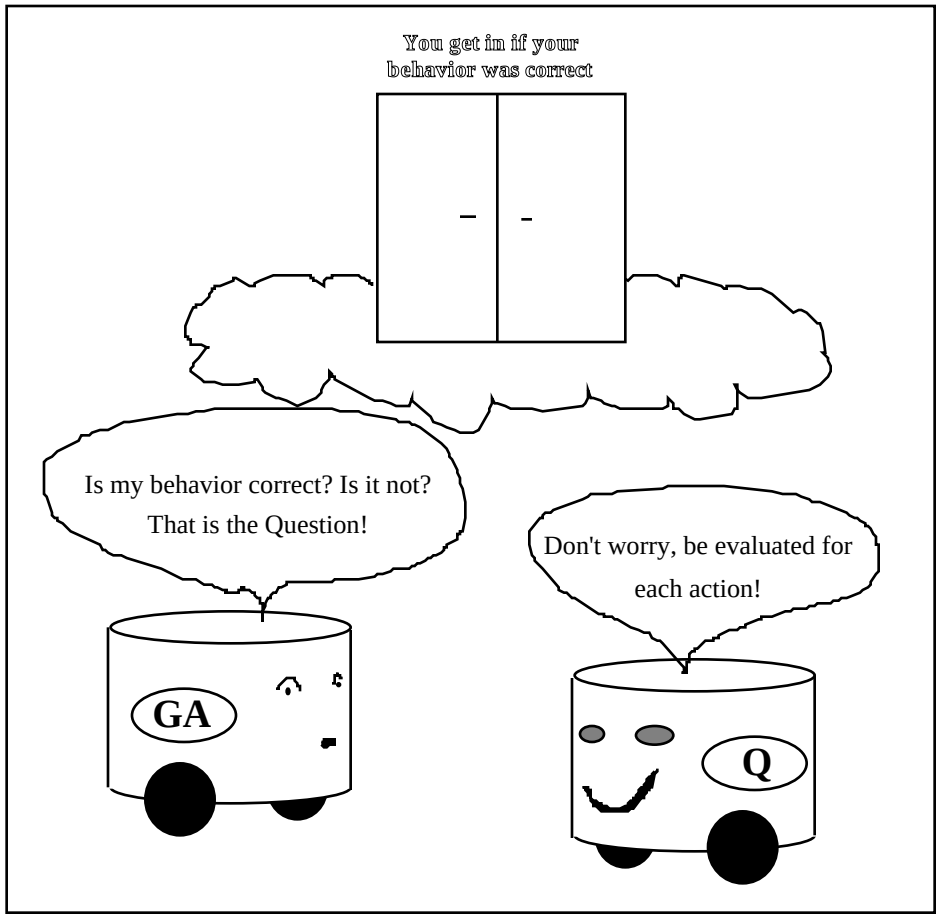
1. Reinforcement learning

Reinforcement learning dates back to the early days of cybernetics and work in statistics, psychology, neuroscience and computer science. In the last five to ten years, it has attracted rapidly increasing interest in the machine learning and artificial intelligence communities. Its promise is beguiling - a way of programming agents by reward and punishment without needing to specify how the task (i.e., behavior) is to be achieved (fig. 14). Reinforcement learning allows, at least in principle, to bypass the problems of building an explicit model of the behavior to be synthesized and its counterpart, a meaningful learning base (supervised learning).



The behavior learned using reinforcement learning contains an implicit model of the robot and its environment. By using reinforcement learning it is not necessary to have examples to build and validate the behavior (fig. 15). The behavior is synthesized by using as a unique source of information a scalar, the so-called reinforcement, which evaluates behavior actions: the agent receives either positive or negative reinforcements according to the utility (i.e., desirability) of the situation entered as a consequence of the performed action. There is no separation between a learning phase and a utilization phase. Also, by using reinforcement learning, only relevant associations between input and output are learned.





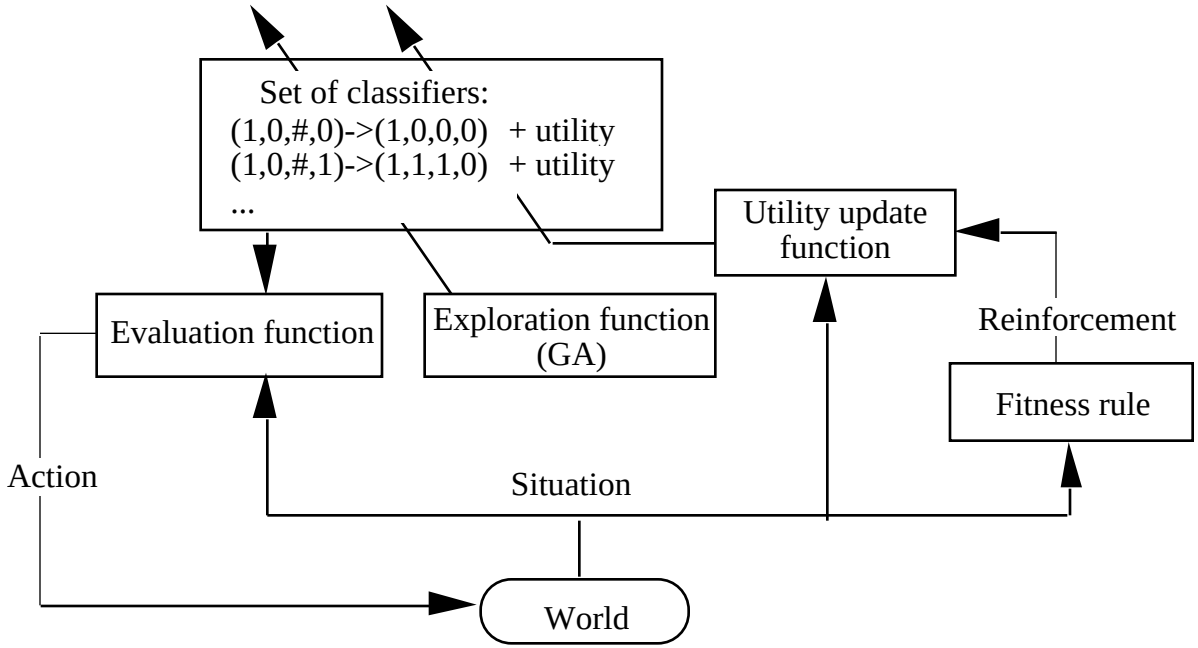
"There are two main strategies (fig. 16) for solving reinforcement learning problems. The first is to search into the space of behaviors in order to find one that performs well in the environment. This approach has been taken by work in genetic algorithms and genetic programming. The second is to use statistical techniques and dynamic programming methods to estimate the utility of taking actions in situations of the world" [Kaelbling 96]. Q-learning is certainly the most used method of dynamic programming.

Genetic Algorithms (Evolutionary Learning)

"Genetic algorithms are generally considered biologically inspired methods. They are inspired by Darwinian evolutionary mechanisms. The basic concept is that individuals within a population which are better adapted to their environment can reproduce more than individuals which are maladapted. A population of agents can thus adapt to its environment in order to survive and reproduce".

The genetic algorithm can be adapted and parametrized to produce an appropriate optimization tool for a single robot behavior synthesis. To this end, the fitness rule (i.e., the reinforcement function), measuring the adaptation of the agent to its environment (i.e., the desired behavior), is carefully written by the experimenter.

Learning classifier system principle: (fig. 17) An exploration function creates new classifiers according to a genetic algorithm recombination of the most useful. The synthesis of the desired behavior involves a population of agents and not a single agent. The evaluation function, which implements a behavior as a set of condition-action rules, or classifiers. Symbols in the condition string belong to {0,1,#}, symbols in the action string belong to {0,1}. # is the don't care identifier, of tremendous importance for generalization. It allows the agent to generalize a certain action policy over a class of environmental situations with an important gain in learning speed by data compression. The update function, which is responsible for the redistribution of the incoming reinforcements to the classifiers. Classically, the algorithm used is the Bucket Brigade algorithm. Every classifier maintains a value, that is representative of the degree of utility of classifiers.



Genetic algorithms and neural networks

The classifier system may be replaced by a neural network. Comparing to classifiers, a neural network implementation allows a continuous mapping of the behavior space (search space): small variations in the structure or the weights of a neural network result in small variations of the robot behavior. A neural implementation modifies the three main components of the learning classifier architecture as follows (fig. 18):

- The evaluation function implements a behavior as a neural network mapping function. Several neural networks may be involved. Situations are the inputs to the neural network which generates actions. The generalization used is the classical neural network generalization.

- The update function associates to each neural network implementation of a behavior a utility value, directly coming from the fitness rule. A learning phase may be added that will modify the neural weights during the life of the agent. In this case, a reinforcement function will be added to the fitness rule to generate immediate reinforcements.

- The genetic algorithm exploration function creates new neural networks, modifying either the network architecture by adding or removing neurons and connections, or only modifying the weights. The genetic algorithm explores the neural space recombining useful neural networks to produce better offsprings.

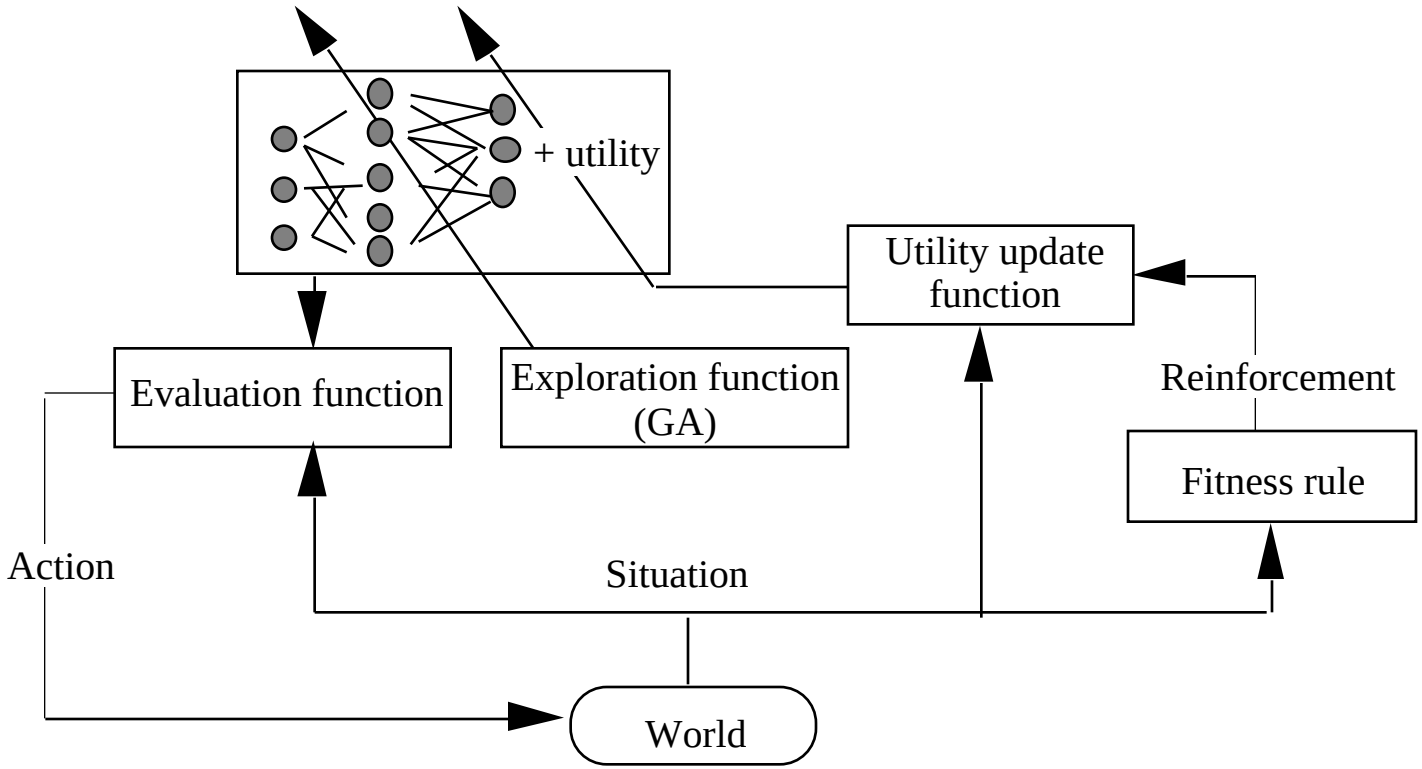
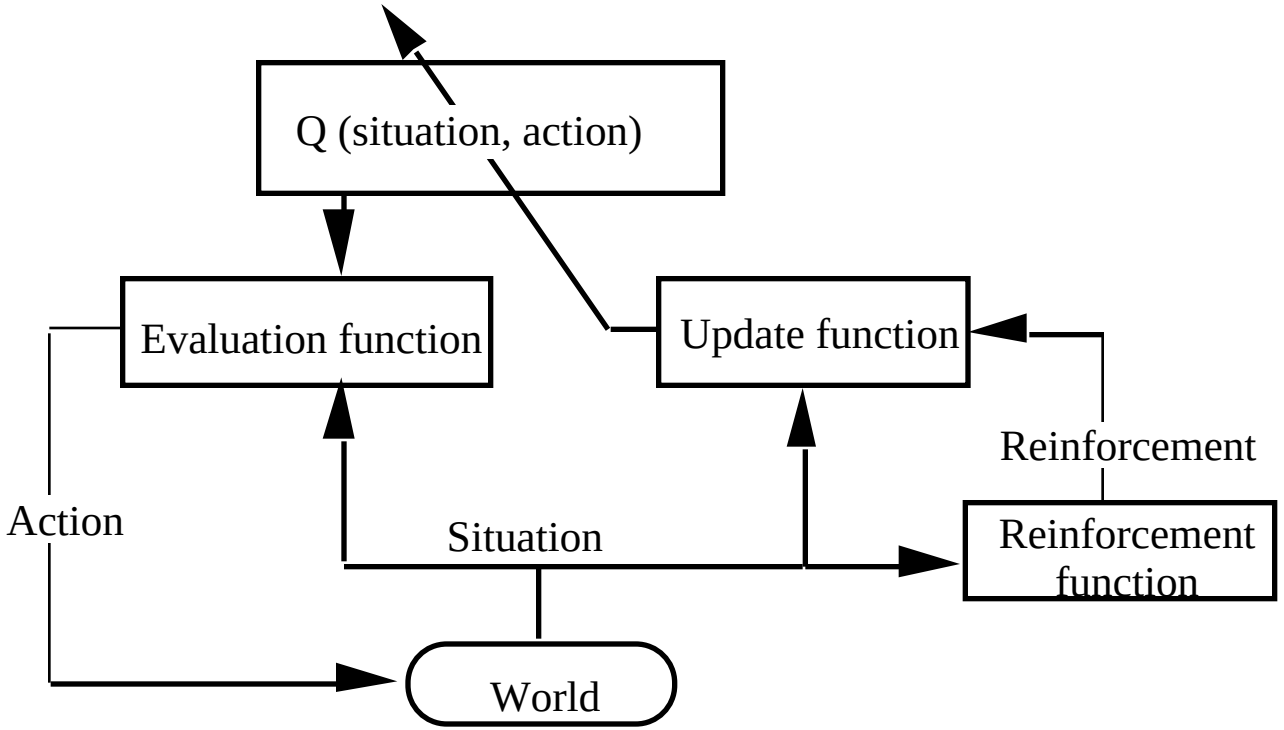


Fig. 18. A neural work implementation of a genetic algorithm architecture.

Limitations of the genetic algorithm approach

Genetic algorithms are slow and only permit to find "close to optimal" behavior, without any guarantee on their convergence and on the quality of the behavior found.



Q-learning stores the utility associated to each situation-action pair (fig. 19). Three different functions are involved: memorization, exploration and updating (fig. 19). Q-learning method functional decomposition. In response to the present situation, an action is chosen by the evaluation function with the help of the robot memory. This action is the one that has the best rewarding probability. After the execution by the robot of the action in the real world, a reinforcement function provides a reinforcement value. This value, a simple qualitative criterion (+1, -1 or 0), is used by the updating function to adjust the reward value (Q) associated to the situation-action pair stored in the robot memory.

Evaluation Function

The Q-Learning algorithm builds a Q function that maps situation-action pairs (i, a) into expected returns r. Q(i, a) is the system's estimate of the return it expects to receive given the fact that it executes action a in situation i. The algorithm uses a lookup table to store the estimated cumulative evaluation Q. All these values represent the internal state (fig. 20 & 21). Non empty cells are the already tried pairs.

r	a_1	a_2	a_3	a_4	a_5	a_6
s_1					+1	
s_2		+1				
s_3		0	0			
s_4			-1		0	
s_5			-1			
s_6	0		0			-1

a

Q	a_1	a_2	a_3	a_4	a_5	a_6
s_1	0	0	0	0	+.7	0
s_2	0	+.8	0	0	0	0
s_3	0	+.4	+.3	0	0	0
s_4	0	0	-1	0	+.8	0
s_5	0	0	-.9	0	0	0
s_6	+.1	0	0	0	0	-.7

b

1. Initialization of the robot memory: for all situation-action pairs, the associated Q value is 0 (i.e., $Q(i, a) = 0$).
2. Repeat :

   1 - Let i be a world situation.

   2 - The evaluation function select the action a to performed:

   $a = \text{Max}(Q(i, a'))$

   where a' represent any possible action. The selection process can be slightly different (stochastic) so as to be able to explore new era of the situation-action space.

   3 - The robot executes the action a in the world. Let r be the reward (r can be null) associated with the execution of the action a in the world.

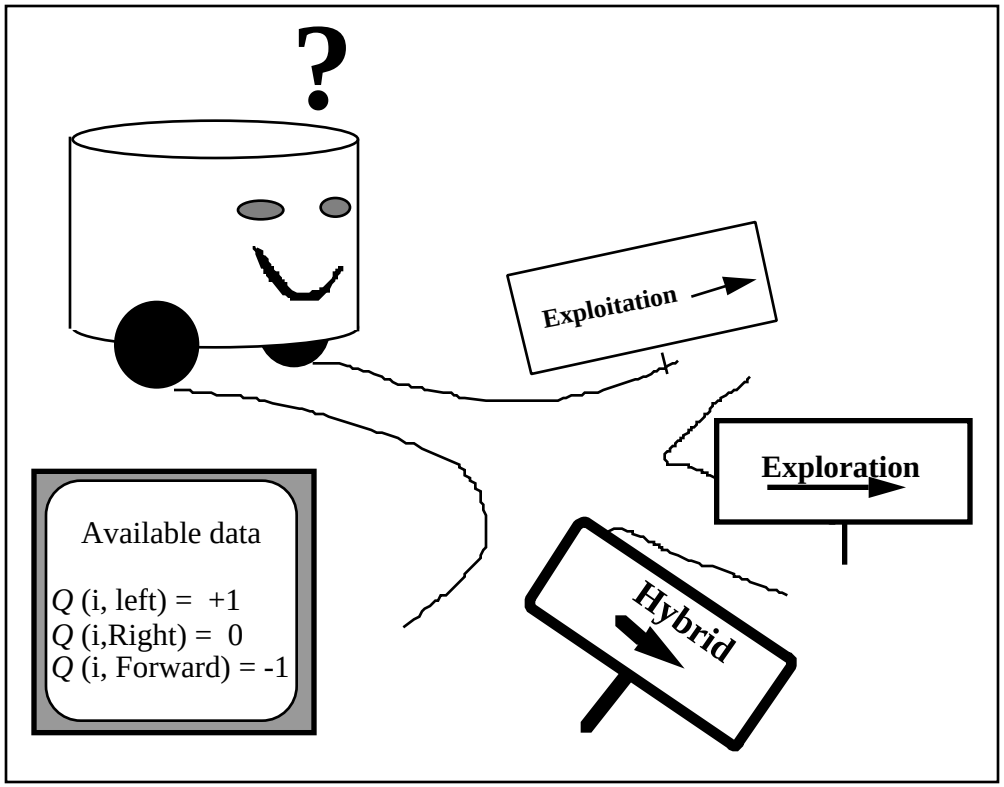
   4 - Update the robot memory:

   $Q_{t+1}(i, a) = Q_t(i, a) + ß(r + g \cdot \text{Max}(Q_t(i', a')) - Q_t(i, a))$. eq. 1

   where i' is the new situation after having carried out the action a in situation i , a' represent any possible action and $0 < ß, g < 1$.

Fig. 21. A general algorithm for the Q-learning method.

One major difference between reinforcement learning and supervised learning is that a reinforcement-learner must explicitly **explore** its environment, i.e., real-time building of the learning base.



If the evaluation function always chooses the actions with the highest estimated payoff, then the flaw will be that early unlucky sampling might indicate that the best action's reward is less than the reward obtained from a suboptimal action. The suboptimal action will always be picked, leaving the true optimal action starved of data and its superiority never discovered. An agent must explore to ameliorate its outcome. A simple exploration strategy is to take the action with the best estimated expected reward by default, but with probability *p*, choose an action at random. *p* value can be large at the beginning to encourage exploration and then slowly decreasing (fig. 22).

Update function

The internal state Q is updated by the following function:

$$Q(i, a)_{new} = Q(i, a)_{old} + ß(r + g.MaxQ(i', a) - Q(i, a)_{old})$$

where i' is the situation after executing a in i, ß and g are constant coefficients, between $0 < ß, g < 1$. The reinforcement at the present time should be equal to the expected returned rewards. The error between the expected value r+ g.MaxQ(i', a) and the current value Q(i,a) must then be minimized.

This updating rule has the effect of propagating a reward associated with a given situation-action to previous pairs of situation-action. It is, in fact, a way to backpropagate delayed rewards (fig. 23).

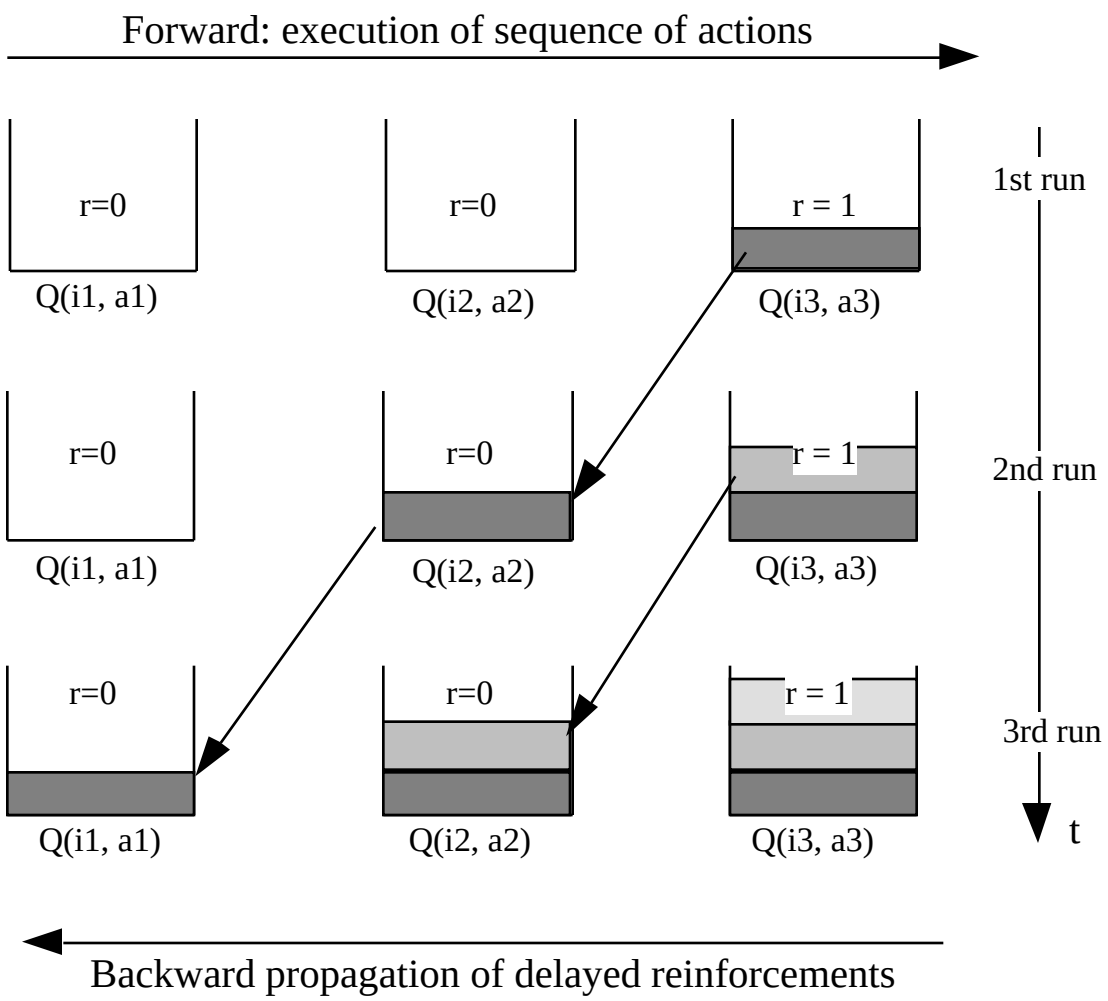
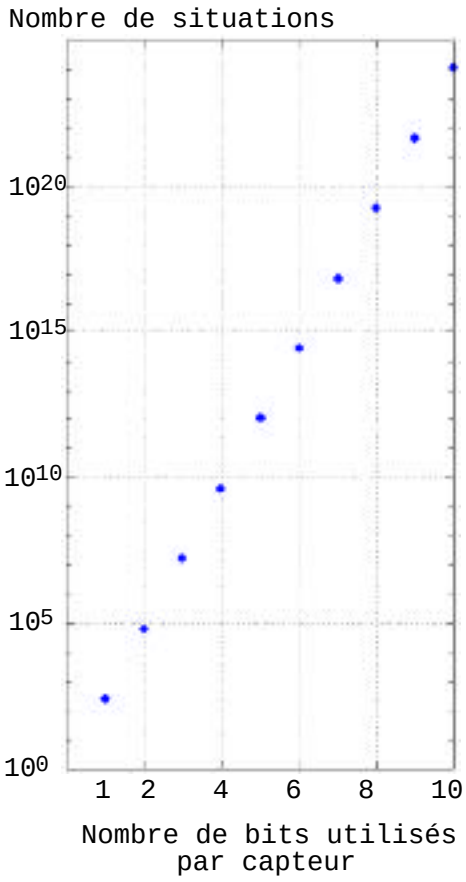


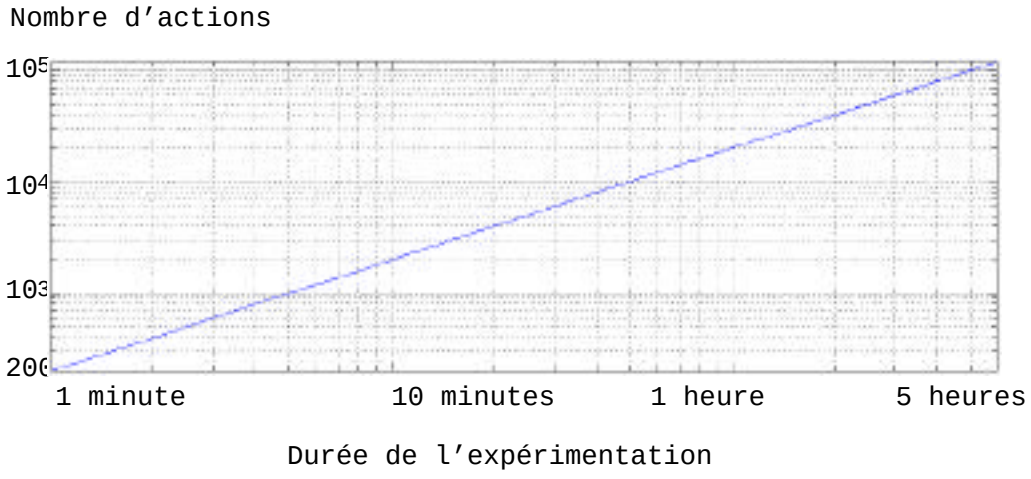
Fig. 23. Example of backward propagation of a delayed reinforcement on 3 runs of the same sequence of situation-action: (i_1,a_1),(i_2,a_2),(i_3,a_3).

Proof of convergence

Recent developments in the theory of reinforcement learning have allowed to prove asymptotic convergence [Dayan 94]. These proofs rely on several assumptions that do not apply to robots facing real-world tasks. In particular, the asymptotic convergence requires a discrete coding of the situation-action pairs (tabular representation) and to try out every action for every situation an infinite number of times. In the real world, the situation-action space is continuous. Thus the robot requires compact representations, such as neural networks, to generalize between similar situation-action pairs and to limit its knowledge (memorization) to relevant parts of the problem only. Experimental results demonstrate that, despite the lack of convergence proofs, reinforcement learning can be successfully applied to real world problems.

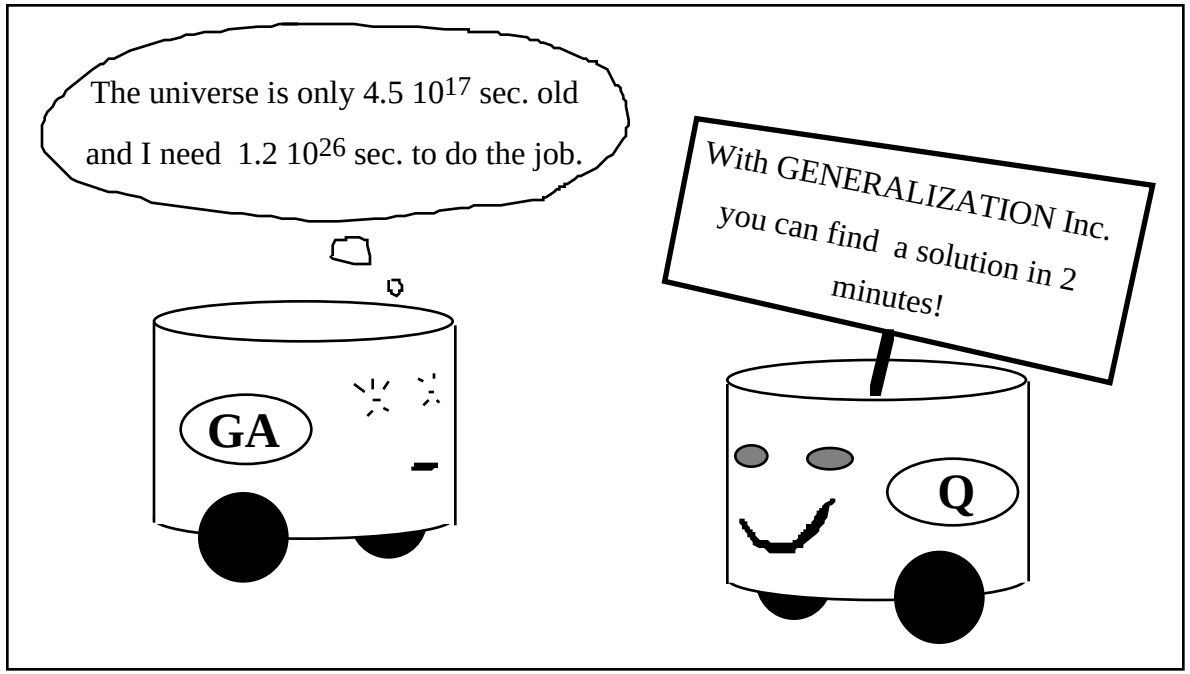


(a)



(b)

Limitations: the situation space is so large (fig. 24a), that combined with all possible actions, an exhaustive exploration (fig. 24b) of all situation-action pairs is impossible, as is also an exhaustive memorization (X and Y are logarithmic scales).



A solution to these limitations is the **generalization** process: the use of experienced situation-action pairs to deal with new unknown situations (fig. 25). Fig. 25. In the case of the mobile robot Khepera, the total number of possible situations is $((2)^{10})^8$, not far from 10^{24}. Combined with 20 speeds per wheel, the size of the situation-action space is $4\ 10^{26}$. Working with a real robot implies mechanical constraints: an action takes approximately 300 ms to be performed. In one minute, a maximum of 200 actions can be executed. There is an incredible small ratio explored situation-action pairs versus unknown situation-action pairs. This problem is called the credit-assignment problem. Generalization is the solution.

***Q-Learning with weighted Hamming distance*[Mahadevan 91]**

The main idea of this refinement is to compute a Hamming distance between the world situation i and similar situations in order to apply the updating function to all of them. Only one action is carried out, but many similar situations are updated using the same reinforcement value. The Hamming distance between any two situations is simply the number of bits that are different between them. Bits can be of different weights. Two situations are distinct if the Hamming distance between them is greater than a fixed threshold (fig. 26). This generalization method is limited to syntactic criteria: it is dependent on the coding of the situations.

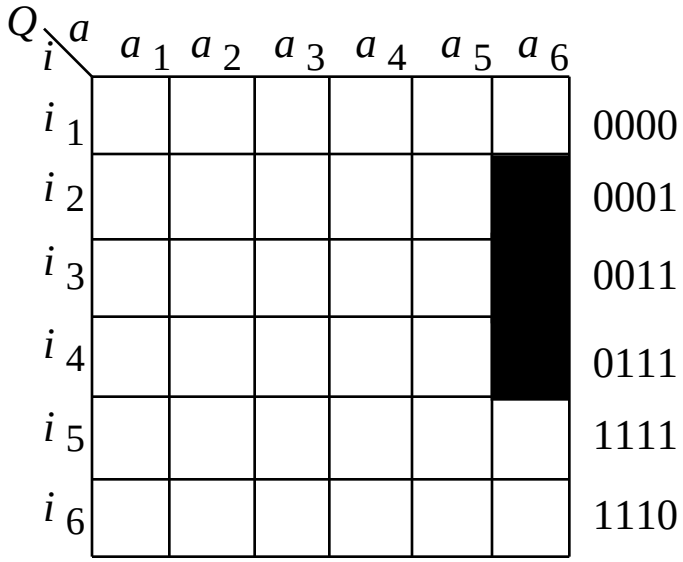


Fig. 26. Generalization using Hamming distance in black. In this example, the world situation is $i3$, the executed action is $a6$, the threshold value for the Hamming distance is 1. Q ($i3$, $a6$) is updated, but also Q ($i2$, $a6$) and Q ($i4$, $a6$).

Q-Learning with statistical clustering

Mahadevan *et al*. proposes an other generalization method less dependent on the coding of the situations: statistical clustering. Here, each action is associated with a set of situations giving information concerning the usefulness of performing the action in a particular class of situations. Clusters are a set of "similar" situation instances that use a given similarity metric. All situations that appear in the same cluster are updated together (fig. 27). Here again, generalization is limited to syntactic criterion.

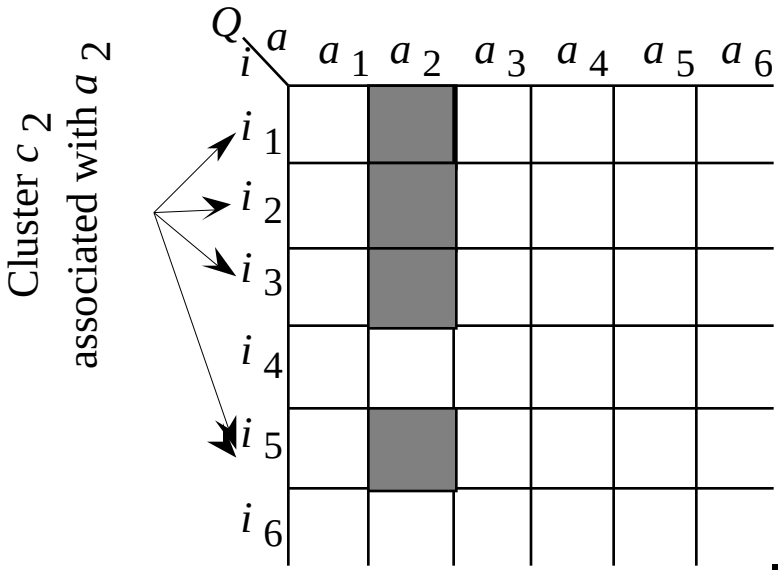


Fig. 27. Generalization using statistical clustering. If one of the situations of the cluster c_2 is the world situation and a_2 has been carried out then all Q values of the cluster c_2 are updated. In this example, the world situation is i_3 and the action carried out is a_2 , then Q (a_2 , i_3) is updated together with Q (a_2 , i_1) ,Q (a_2 , i_2) and Q (a_2 , i_5) .

Conclusion: Reinforcement learning is justified if it is easier to implement the reinforcement function than the desired behavior, or if the behavior generated by the agent presents desirable emergent properties (like generalization, robustness, redundancy, adaptability) which cannot be directly built. This last reason is certainly the best motivation for the use of reinforcement learning in autonomous robotics.

"Pure" reinforcement learning systems receive sparse reinforcement. In other words, they are rewarded or punished only in special occasions. The poverty of this information, a delayed scalar, and related problems, like how to distribute the reinforcement to the different parts of the system which contributed to the achievements of a goal (or a bad state) causes one of the major drawbacks of reinforcement learning: its long convergence time.

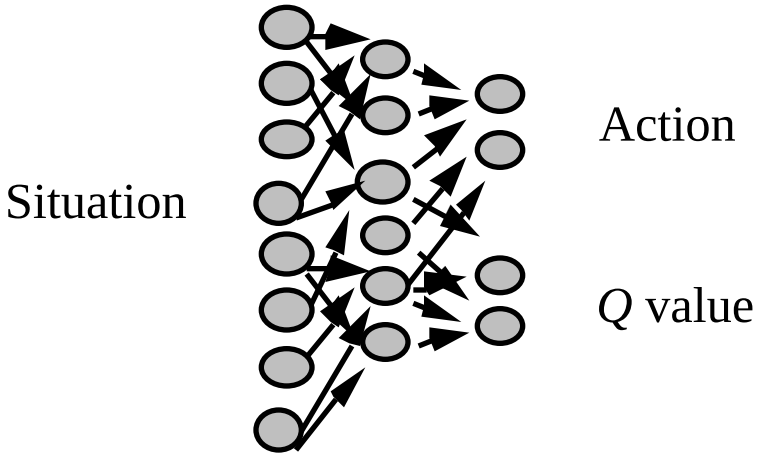
3. Q-learning Neural Networks Implementations (1992)

The first neural implementation of reinforcement learning occurred in the beginning of the eighties [Barto 85]. Neural Q-learning implementations were proposed in the nineties [Lin 92]. A neural implementation seems to offer many advantages: quality of the generalization and limited memory requirement for storing the knowledge. The memorization function uses the weight set of the neural network. The memory size required by the system to store the knowledge is defined, a priori, by the number of connections of the network. It is independent of the number of explored situation-action pairs.

Multilayer perceptron implementations

Ideal implementation

The ideal neural implementation will provide, in a given situation, the best action to undertake and its associated *Q* value (fig. 28). This ideal model only provides one output, i.e., an action and a *Q* value per situation. This action should be the best available action in the situation. How can we be sure of the fact?



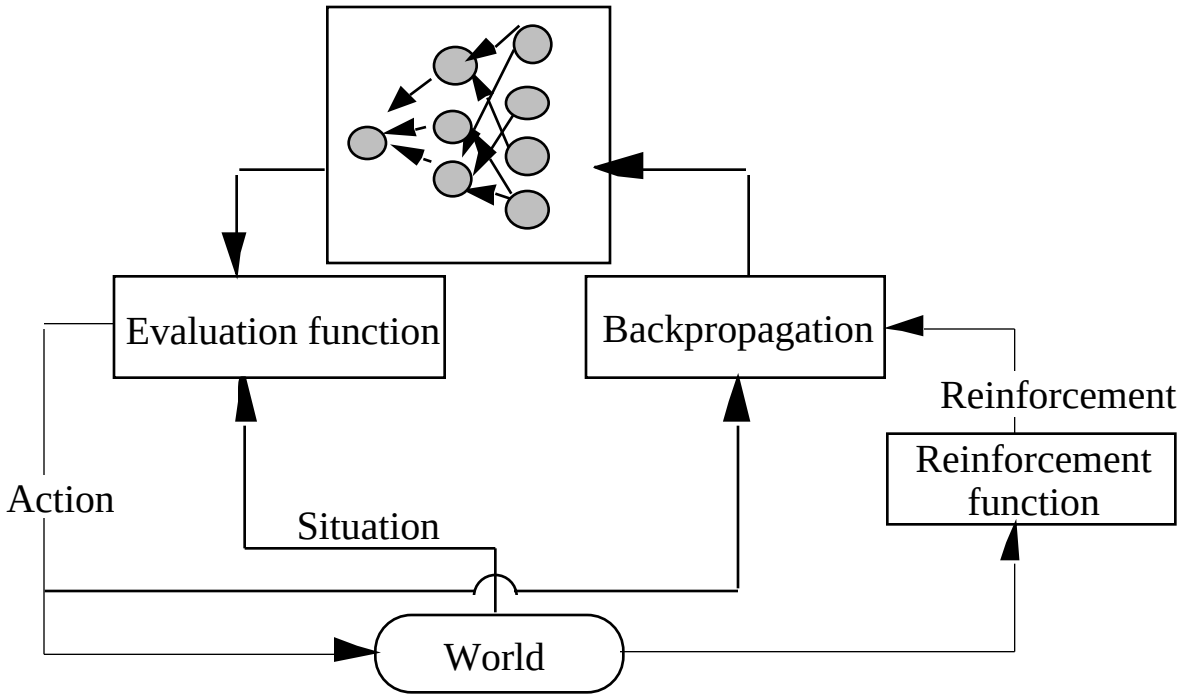


Fig. 29. The direct implementation of the ideal neural network implementation of the Q-learning with a multilayer backpropagation network. The updating function is a weight modification algorithm, here the well-known gradient error backpropagation algorithm [Rumelhart 86]. An error signal on the output neurons must therefore be defined for each output neuron. How can a quantitative error signal be defined when the only available information is of qualitative nature? The definition of this error is restricted to simple cases where only two actions are possible.

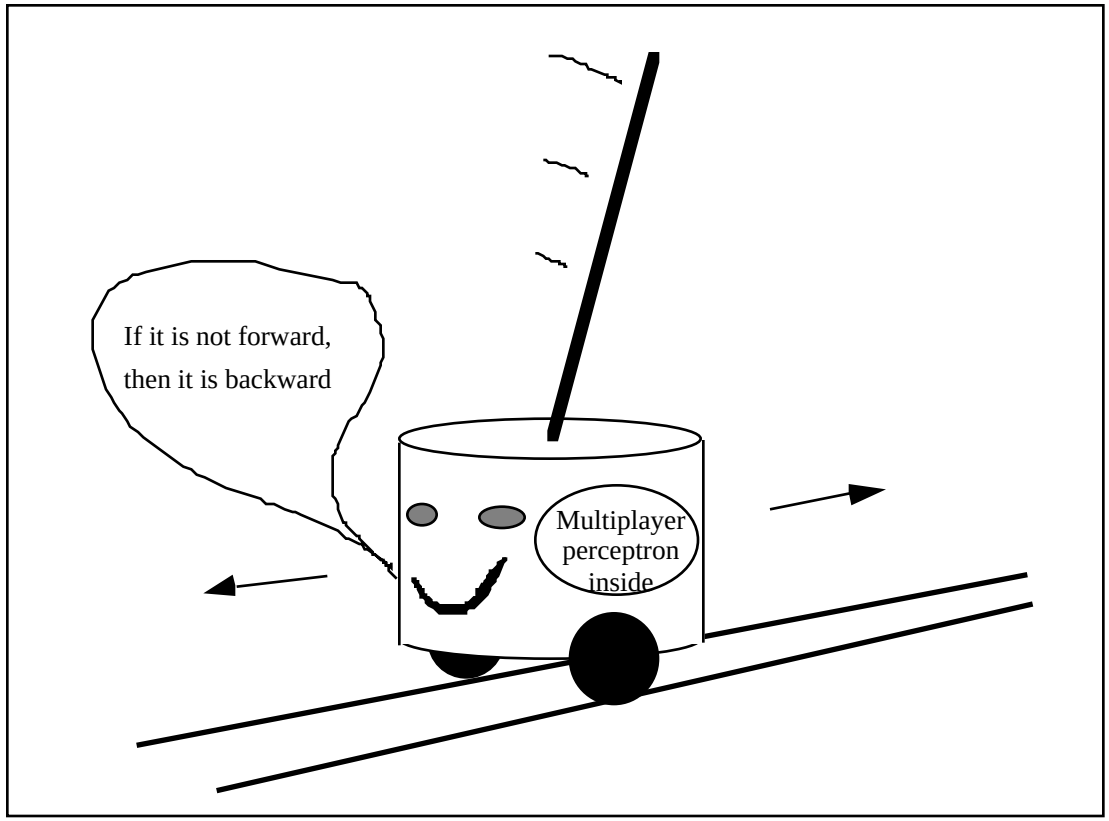


Fig. 30. One of the first application used to demonstrate the power of the neural implementation of the Q-learning was the inverse pendulum with only two actions (moving the cart left or right). In this case, it is easy to deduce from the reinforcement signal the desired output value (fig. 30). For applications involving many possible actions, like mobile robotics (e.g., Khepera allows 400 different actions), dealing with negative rewards is more difficult and implicates modifications of the ideal implementation.

Lin proposes the **QCON** model: a multilayer perceptron implementation of the Q-learning algorithm which characteristic is to have only one output neuron. There are as many QCON networks as there are actions (fig. 31). It is impossible to generalize across actions.

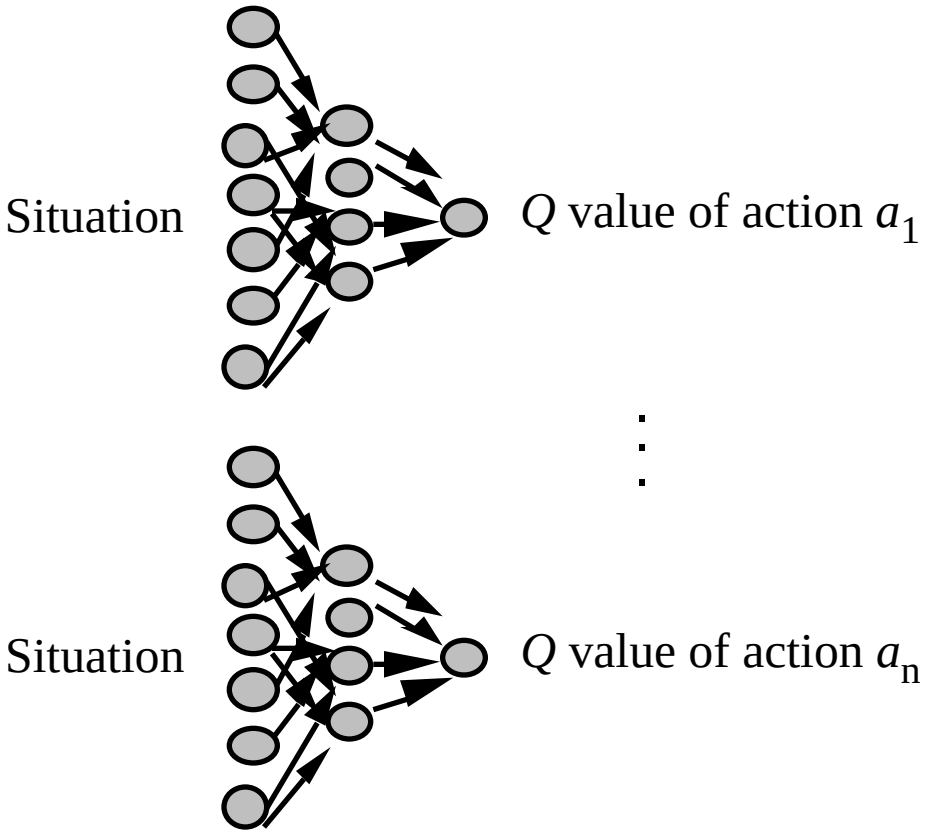


Fig. 31. The QCON model: only one output neuron per multilayer neural network. Each network is associated to a unique action and the output is considered as the Q value associated to the situation-action pair. There are as many QCON networks as there are actions. Generalization across situation-action pairs is impossible. Learning concerns only one network per iteration.

Unmapping in multilayer backpropagation neural networks

Generalizing across actions implies having an implementation architecture composed of only one network so as to be able to use every reinforcement signals to update the network. Moreover, generalizing across actions implies that the output layer codes actions and not *Q* values. Since the neural model is still a multilayer perceptron with a backpropagation learning algorithm, an error must be defined on the output layer. There is no problem when the reinforcement signal is positive, the proposed action is the desired one. But, with negative reinforcements, how can a desired action be chosen? At least, even if it is not possible to find the right situation-action association, a mechanism must be built that will unmap the wrong situation-action association proposed by the network.

The first idea that comes to mind when the number of possible actions increases is to learn the inverse mapping [Barto 85]. The problem then is to determine among all those left which action is the most in opposition. Because the output of the network is numerical, we can change the sign of the output values. However, this is a harmful way of unlearning. Nobody knows what has been deleted. Representation on a neural network is distributed, so it is not possible to delete only one association (or situation-action pair) without interfering with the rest of the learned knowledge. Moreover, a negative reinforcement does not always mean that the error is important, and to learn the inverse action can be defective. With the same goal in mind, Ackley (1991) proposes the use of the complement of the generated output.

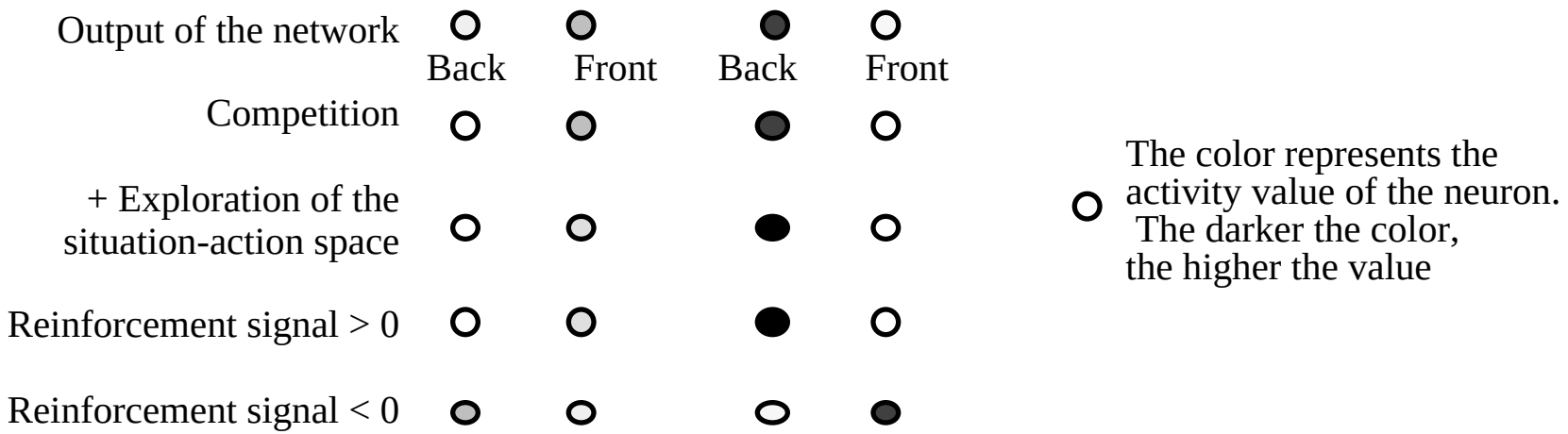
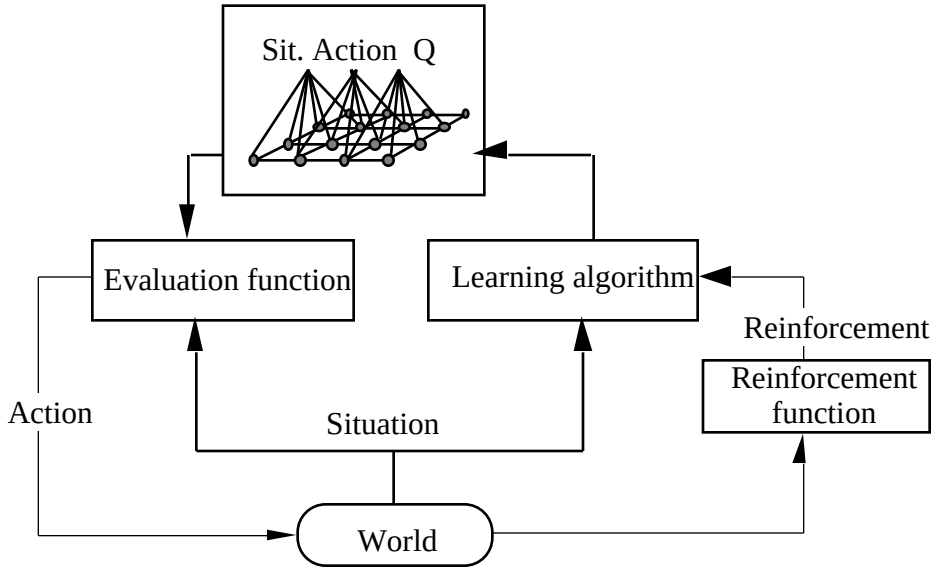


Fig. 32. An other method is to have two output neurons per actuator and establish a competition among them. The idea is to consider the output value as the network confidence in its proposal. The exploration process can modify the proposal so to explore the situation-action space. If the reinforcement signal is positive then the error is equal to the value added by the exploration process. There is no error for the other neuron of the pair. If the reinforcement signal is negative then values in each pair of neurons are exchanged. Results are completely dependent on the nature of the application and are not yet satisfactory.

Q-KOHON: a self-organizing map implementation



Experiments show that the competitive multilayer perceptron implementation learns faster than the other reviewed implementations: generalization is better. This is due to the localized coding on the output layer: one output neuron for each actuator; and also to the competition between output neurons. Therefore, the implementation of the Q-learning with a neural network model, e.g., the self-organizing map (SOM), that is completely dedicated to these two points must be very powerful. Coding on a SOM is localized. Each neuron represents a particular class (or cluster) of the inputs. Competition occurs between all the neurons of the map. During the learning phase, the neurons of the SOM approximate the probability density function of the inputs. The inputs are situation, action and the associated Q value (fig. 33). The learning phase associates to each neuron of the map a situation-action pair plus its Q-value. It is a method of state grouping involving syntactic similarity and locality [McCallum 95]. The number of neurons equals the number of stored associations. The neighborhood property of the Kohonen map allows to generalize across similar situation-action pairs.

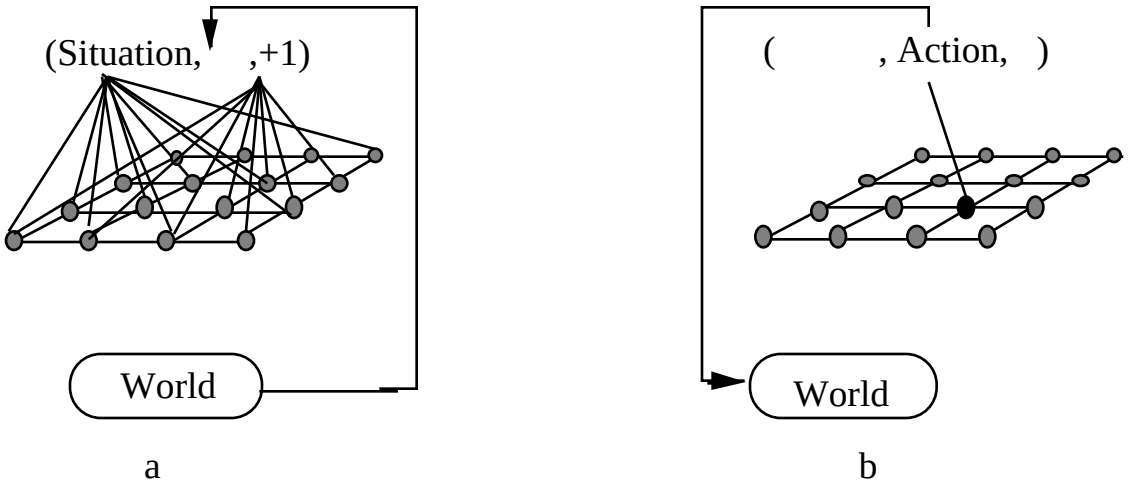
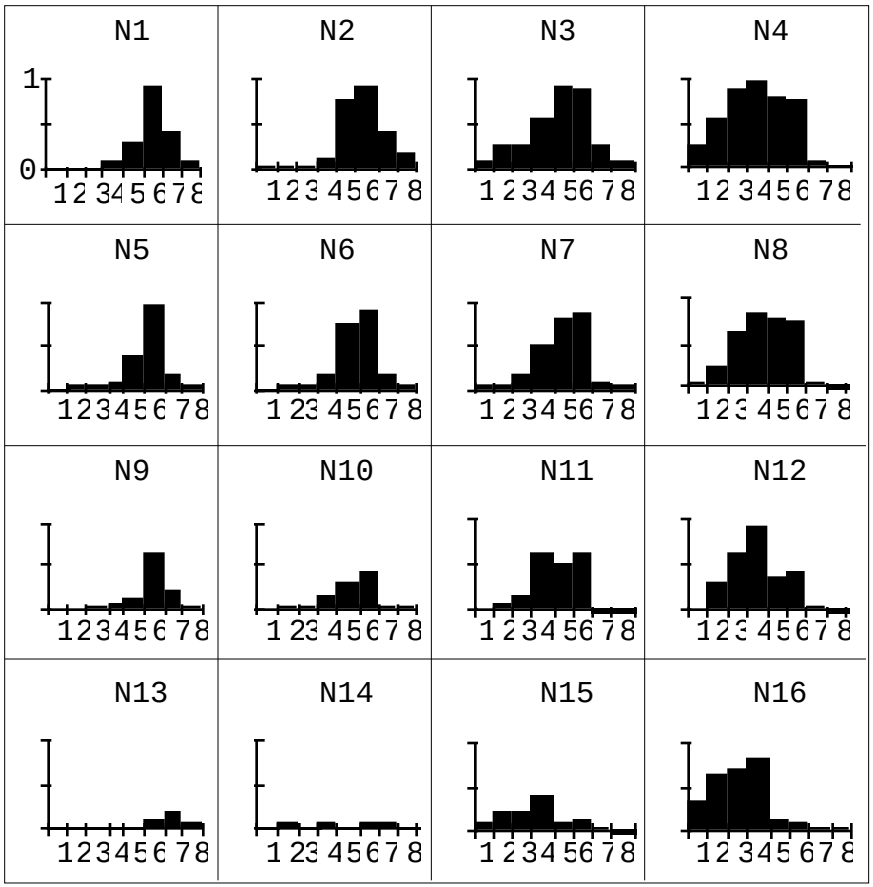


Fig. 34. Selection of the best action to perform in the world situation. The Kohonen map is used as an associative memory: information is probed with part of it.

a/ The world situation and a *Q* value of +1 are given as inputs.

b/ The answer is a selected neuron which weights give situation, *Q* value and the associated action.

The learning algorithm updates the *Q* value weight and, also, the situation and action weights. The neuron corresponding to the situation and the action effectively performed is selected. The distance used is different from the exploration process. It includes the situation and action vectors, but nothing concerning the *Q* value. Together with the selected neuron, the four neighbors are also updated. The learning coefficient is 0.9 for the selected neuron and 0.5 for the neighborhood. During the learning, the influence on the neighbors decreases inversely proportionally to the number of iterations.



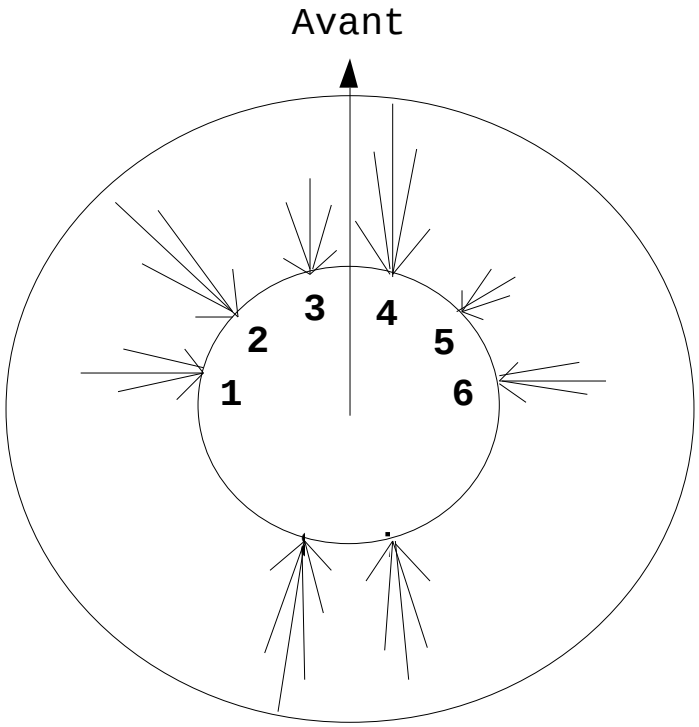
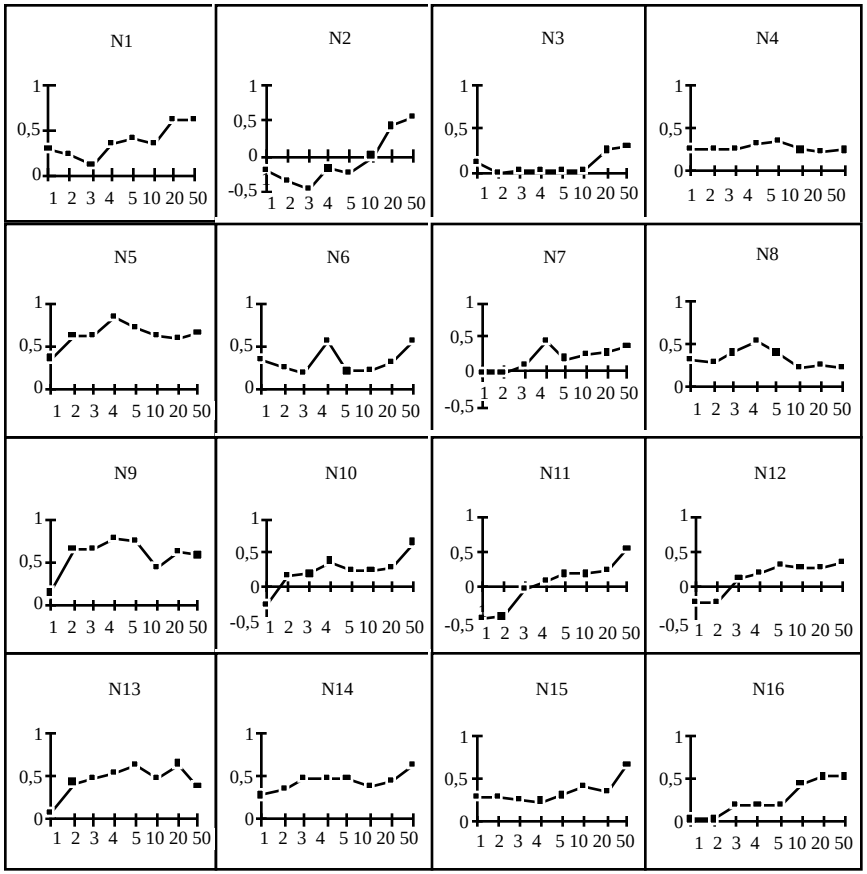


Fig. 35. Visualization of the eight weights linked to the situation input for each neuron of the map after 200 learning iterations in a task of obstacle avoidance behavior synthesis. The highest the value of the sensor, the more sensitive the corresponding neuron to an obstacle. These diagrams represent the sixteen shapes of obstacles used for the classification. Each class is associated to an appropriate action.

The properties of the self-organizing map allow to predict that, if a correct behavior is learned (i.e., only positive rewards are experienced), then all neurons will code positive *Q* values.



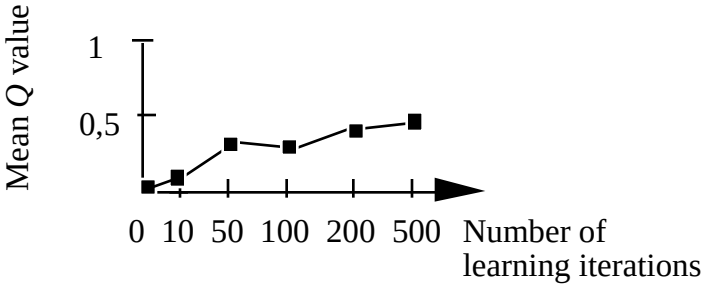


Fig. 36. Evolution of the *Q* value associated with each neuron (or class of situation-action represented by a neuron). The *Q* values, starting from 0.0, converge to positive values for all neurons, demonstrating that the learned behavior is rewarding. At the end of the learning phase, only situation-action pairs giving positive rewards are stored in the map. The number of iterations has to be multiplied by 10.

Comparison:*Obstacle avoidance behavior synthesis*

We run several experiments to be able to compare the different implementations of the Q-learning. In all experiments, Khepera uses a random number generator to control the exploration process. The randomness decreases inversely proportionally to the number of iterations. It is a crude strategy for active exploration, but sufficient for our experiments.

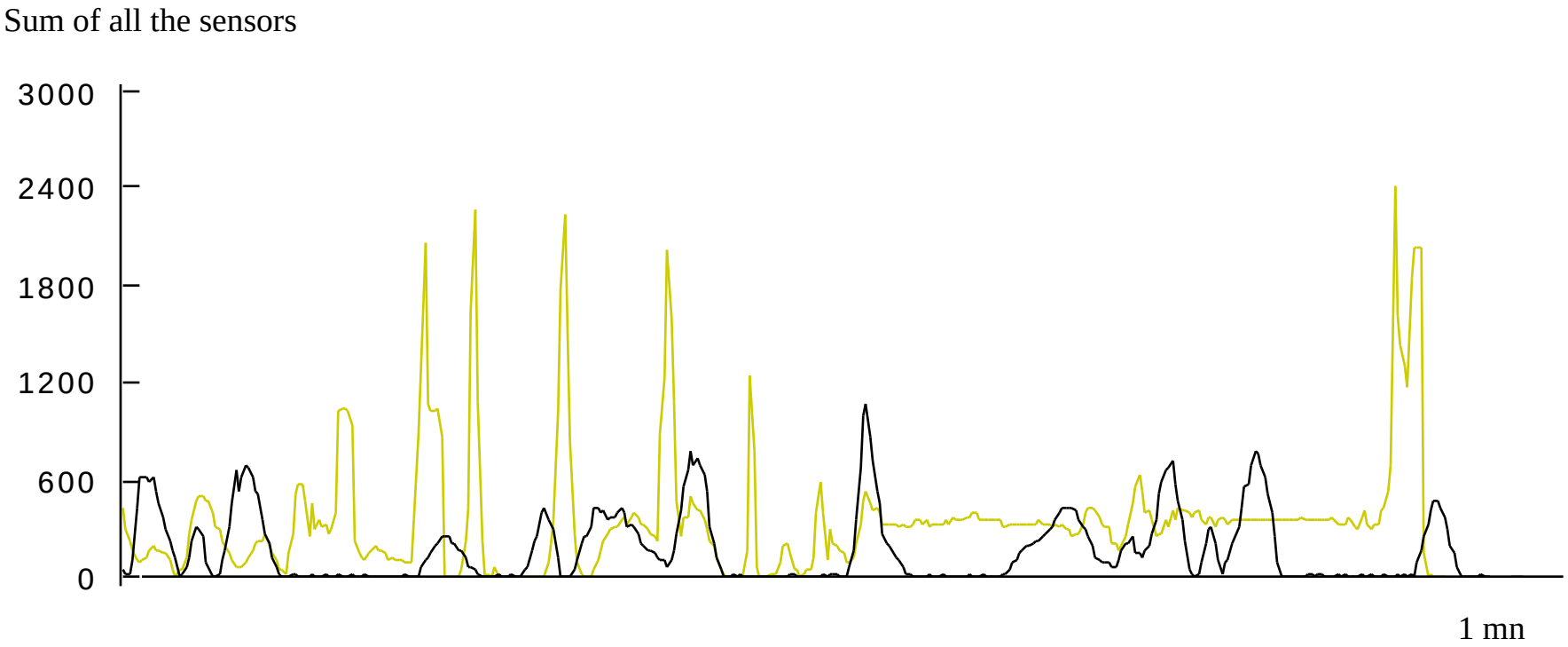


Fig. 37. Sum of all the sensor values after the learning of an obstacle avoidance behavior with the mobile robot Khepera. This one-minute graph (in black color) is obtained with a self-organizing map implementation of the Q-learning. It is also a good presentation of the performances shown by all the other implementations. It is interesting to compare this graph with the graph obtained for the Braitenberg implementation (in gray) of an obstacle avoidance behavior. As we see, the Q-learned solution avoids obstacle in a better way than the algorithmic solution. However, the robot average speed with a Braitenberg implementation is faster than with a Q-learning implementation: it is easier to avoid obstacle slowly.

Objective criterion: distance to obstacle (an action is considered correct if it does not generate a collision).

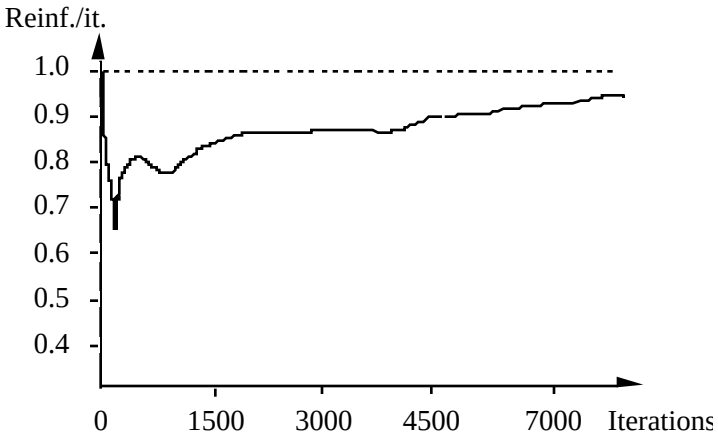


Fig. 38. Cumulative reward over time of the competitive multilayer perceptron implementation of the Q-learning in a task of an obstacle avoidance behavior synthesis. 2000 iterations are necessary to learn a correct behavior. The architecture of the neural network is composed of an input layer of 8 neurons, a hidden layer of 4 neurons and an output layer of 4 neurons. This graph is the mean of five successive experiments.

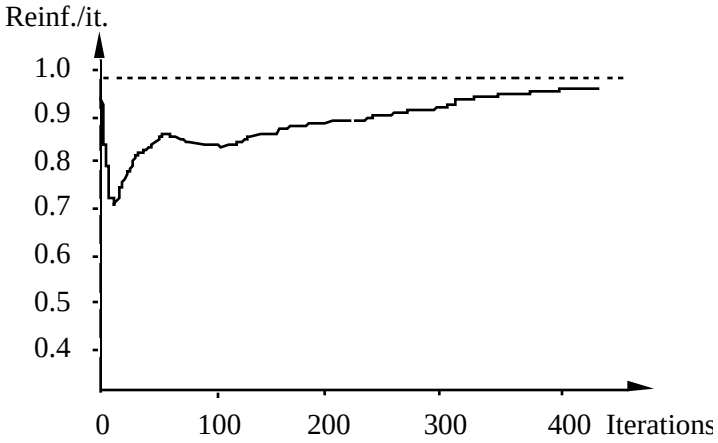


Fig. 39. Cumulative reward over time of the self-organizing map implementation of the Q-learning in a task of an obstacle avoidance behavior synthesis. 200 iterations are sufficient to learn a correct behavior. There are sixteen neurons in the map (176 connections (11 x 16)). This graph is the mean of five successive experiments.

A comparison with the other implementations of the Q-learning has been done. Results displayed on fig. 40 show that the self-organizing map Q-learning implementation requires less memory and learns faster than all the others (by a factor of 100).

	Q-learning	+ Hamming	+ clusterisation	Dyna-Q	Competitive MLP	Q-KOHON
Memory (float)	6400	6400	1.6 M	6400	56	176
# iterations	7500	3500	4000	6000	2000	500
Time	55 mn	25 mn	30 mn	45 mn	8 mn	2 mn

Fig. 40. Comparison of several implementations of Q-learning on a task of learning an obstacle avoidance behavior. The self-organizing map Q-learning implementation (right) requires less memory and learns faster than all the others. The increase in performance (learning time) is superior to 100 when compared to the basic Q-learning implementation (left).

Knowledge Incorporation is *a priori* knowledge incorporation. Biases are devised to be *ad-hoc* to the application, and therefore can take extremely different forms. Thrun and Pratt's taxonomy [Thrun et al., 1998] of biases, in the context of learning to learn, is built in respect to the way the biases implement their actions: partitioning (the exploration, the search space, or the target behavior), constraining (the exploration, the exploitation, or the search space), and other approaches. Their taxonomy criterion reflects the implementation of the biases. We prefer to use a taxonomy criterion which takes into account the target of the bias action---where does the bias apply? We have extracted from the literature three classes:

- one that acts at the search level to improve exploration,
- a second class that intends to reduce the *SSS*,
- and a third class that reduces the learning base size by simplifying the target behavior.

Exploration biases are intended to provide a guidance for the exploration of the search space, improving the quality of the learning base by selecting more significant samples. For example, Millán [Millán, 1996] provides the robot with a set of *reflexes* (Fig. 41), which are used every time the evaluation function (a connectionist controller) does not find an input neuron matching the current situation. It is expected that the neural network gets control more often as the robot explores the environment, and increases the performance of the policy. The connectionist controller is tuned through reinforcement learning.

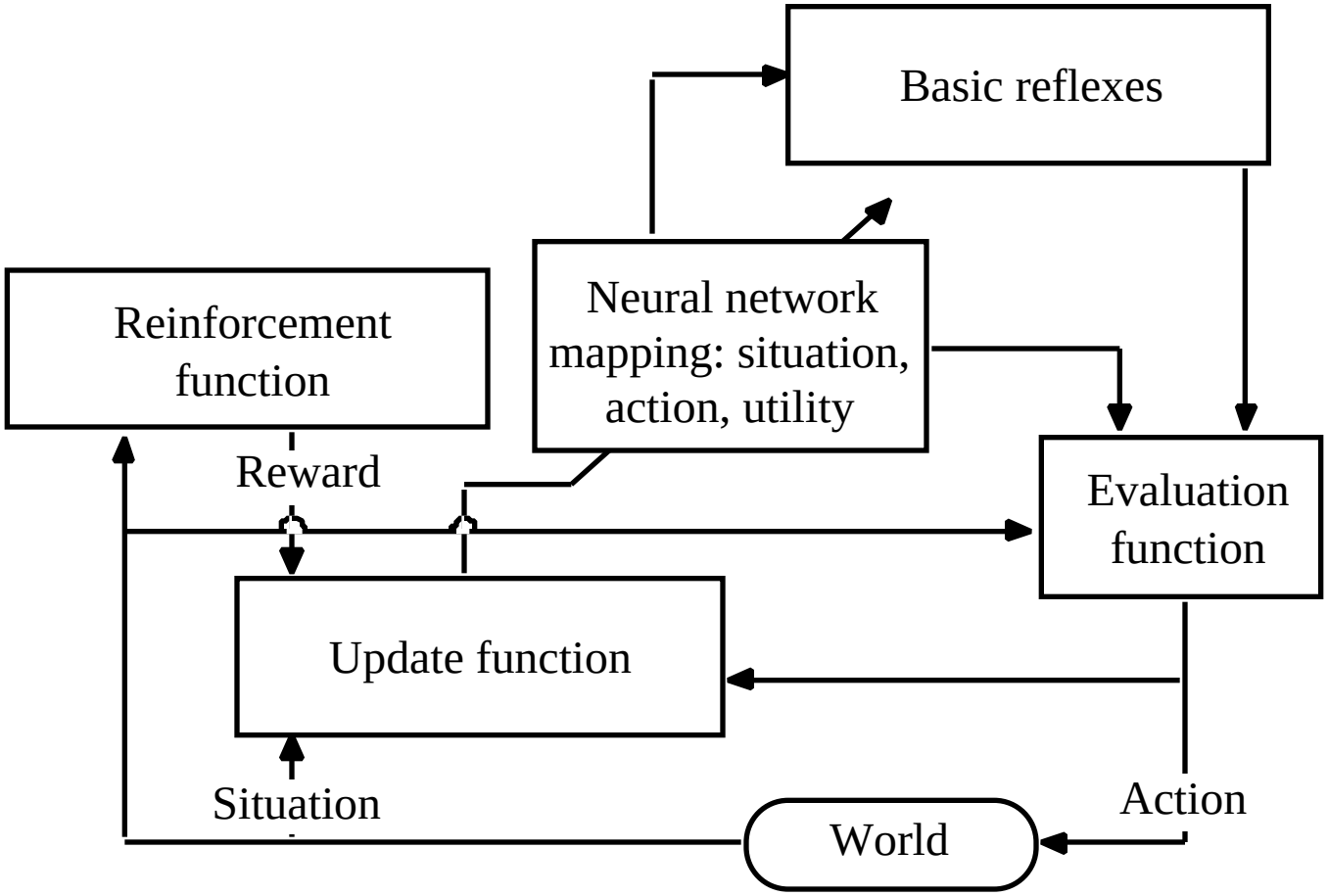


Fig. 41. Reflexes provide a guidance for the exploration of the search space, improving the quality of the learning base by selecting more significant samples.

Bias validity

Improving the quality of the learning base through the use of biases certainly speeds up the learning of the behavior. However, one can wonder at the quality of the final solution. There is no question that the synthesized behavior will conform the learning base, and that the learning base conforms the biases; but the question is how far should we trust the biases? Using learning, we are assuming that our ability to accurately model the environment is not perfect. Therefore, biases which are representations of our knowledge about the environment cannot be considered 100% accurate. This is the reason why a learning phase is necessary to correctly adjust the behavior achieved as a result of the bias use.

However, it must be emphasized that the involvement of biases greatly reduces the learning correction capacities. The generalization is confined to the biased region of the search space. Let us take, as an example, the well-known Braitenberg's obstacle avoidance behavior (Fig. 42) [Braitenberg , 1984] as a bias. Then, we observe the following limitations on the completion and the correction capacities.

Theoritical

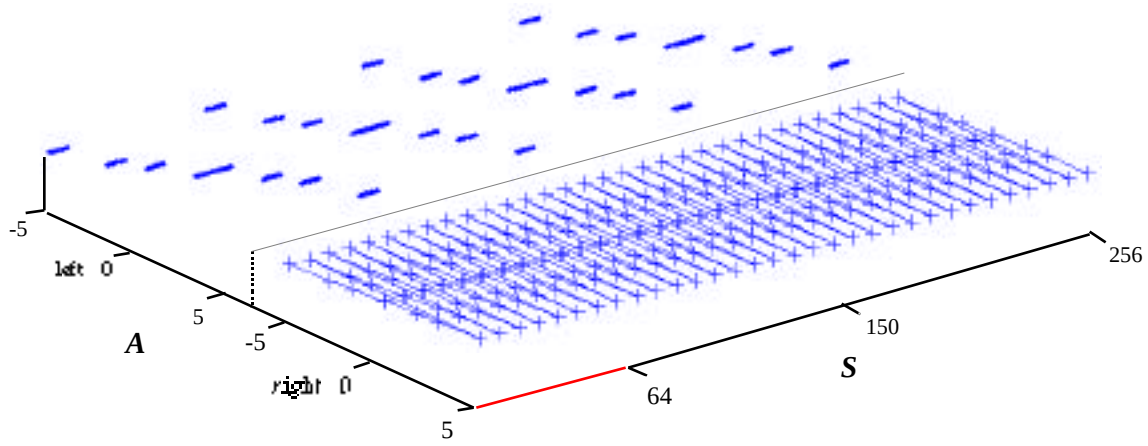


Fig. 42. The behavior of the Braitenberg's obstacle avoidance vehicle. The complicated shape of the mapping for the right speed is due to the impossibility of achieving a continuous representation for the situations along one axis.

Explored

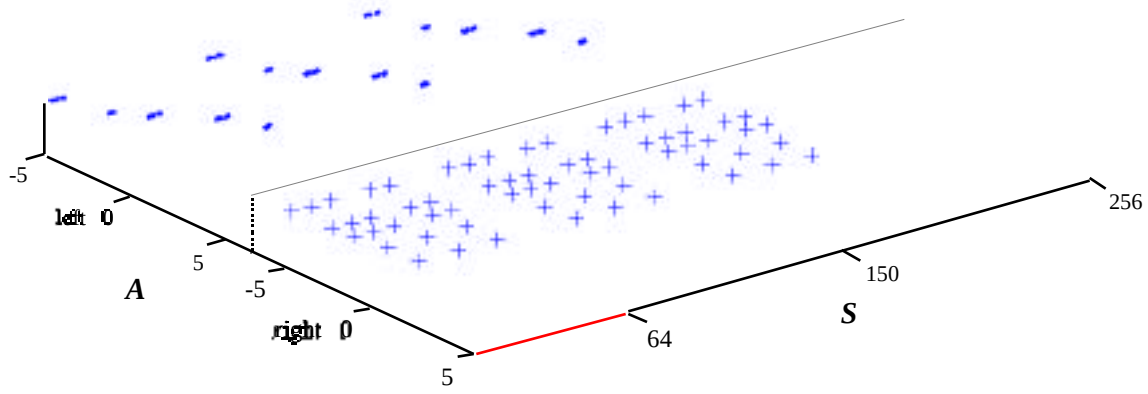


Fig. 43. Situation-action pairs explored using Braitenberg's obstacle avoidance algorithm as a set of reflexes. Situations corresponding to close obstacles are missing and will not allow the learning of an efficient behavior.

A solution

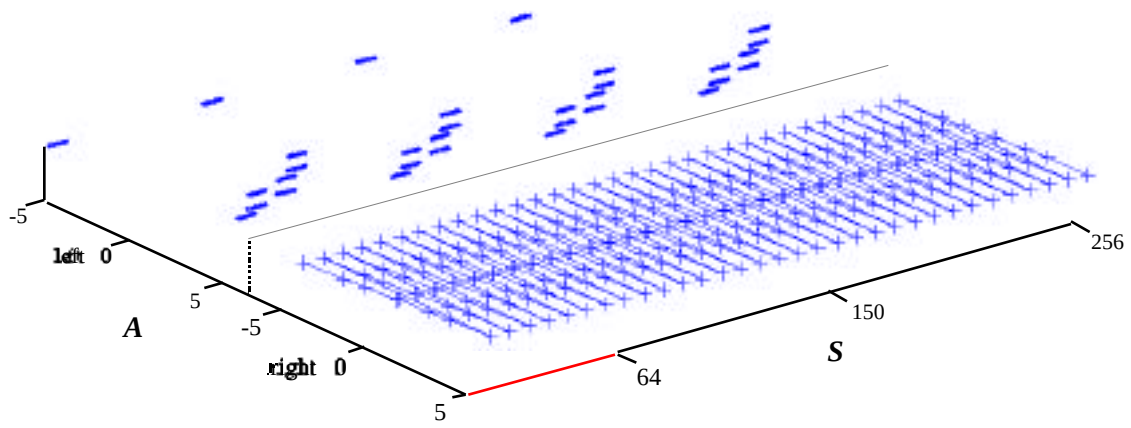
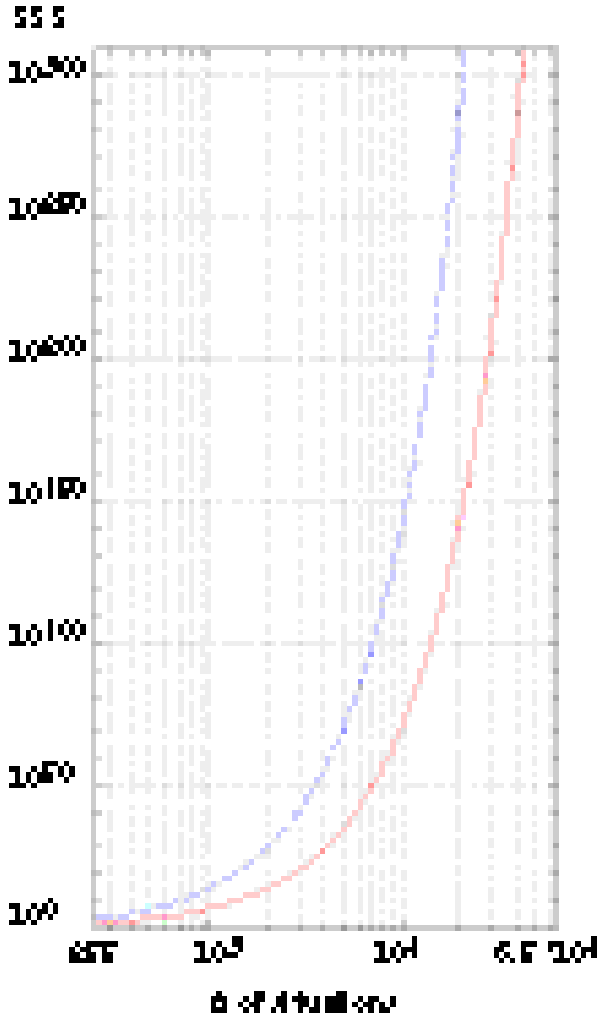


Fig. 44. A possible solution to the "getting stuck in corners" problem. There is no possibility of reaching this policy starting from the biased mapping (fig. 42), due to the intrinsic limited ranges of the exploration and generalization processes (fig. 43).

We see that this mapping is extremely different from the mapping generated by the Braitenberg's bias. There is no chance to obtain this solution using the Braitenberg's obstacle avoidance bias. In this section, we have reported on the non-optimality of the biases, shown that the intrinsic limitations of the completion and correction capabilities may impede the learning of an optimal solution.

Search space size reduction

Search space size reduction biases are intended to reduce the *SSS* either by a decomposition of the task into subgoals, or by the reduction of the number of sensors and actuators. For example [Kalmar et al., 1998] control the dimensionality of their problem by *decomposing the task* (a small mobile robot moving and grasping a ball) into subgoals. On the other hand, Mahadevan [Mahadevan, 1998] propose to *decompose the sensory representation space* so as to accelerate the learning. The effect on the *SSS* is illustrated in Fig. 45 for a decrease by a factor 5 of the number of actions (equivalent to a decrease of each actuator sensitivity by a factor of 2.24). The upper curve is the same as in fig. 24b, the axes have logarithmic scales.



As can be seen in fig. 45, even a small decrease of the number of actions (25 -> 5) has an important impact on the *SSS*, but the *SSS* is nevertheless huge. It is certainly possible to increase the constraints on the sensory information and actions, like in Mataric's work [Mataric, 1997], so as to allow a complete exploration of the search space. However, a large search space provides the robot with an important degree of flexibility that allows the learning process to invent useful solutions (i.e., not explicitly introduce by the human operator), in a similar way as the one pointed out by Mitchell [Mitchell, 1997] in the connectionist domain.

Redundancy in the sensory perception, which could appear as an objective reason for applying sensory perception reduction, has no effect on the effective *SSS*: redundant information does not increase the *SSS* of the potential behaviors. It only affects the theoretical computation of the *SSS*. Also, one of the basic interests in robot learning comes from the possibility to be able to adapt to sensor failures, by using redundant information. Moreover, every sensor information reduction decreases the expressivity of the behavior solutions. Therefore, we can conclude that *SSS* reduction, through the use of sensory reduction, has to be drastic to be of any help, in which case the learning advantages are jeopardized.

Reducing the number of required samples by reducing the behavior complexity

Sample requirement reduction biases are intended to reduce the number of samples required to achieve the learning by reducing the complexity of the behavior solution. This process is different in nature from the decomposition of the task into subgoals (*SSS* reduction biases). The main idea behind Dorigo's s*haping* [Dorigo et al., 1998] is not to reduce the search space, but the expression of the target behavior. A simple behavior requires fewer learning samples to be learned than a more complex one. Then, when the target behavior is correctly synthesized, a more complex behavior becomes the new target. The same idea is used in *Learning from Easy Missions* (or LEM) [Asada et al., 1996]. There is no way today to have a quantitative measure of the complexity of the target behavior. The Vapnik-Chervonenkis Dimension (VC-dim) requires noise-free data, supervised learning and a fixed distribution between exploration and exploitation; application of reinforcement learning to robotics does not fulfill these conditions. In Fig. 46, we represent the complexity of a behavior as the minimum number of points necessary for describing the low-level behaviors (we consider each sensor individually). One point is required for a constant action output, 2 points for a segment, 3 points for two segments, and so forth., until the maximum represented here of 6 points for 5 segments. We compute the number of samples required for the learning as the number of potential combinations (equal to the number of points power the number of sensors).

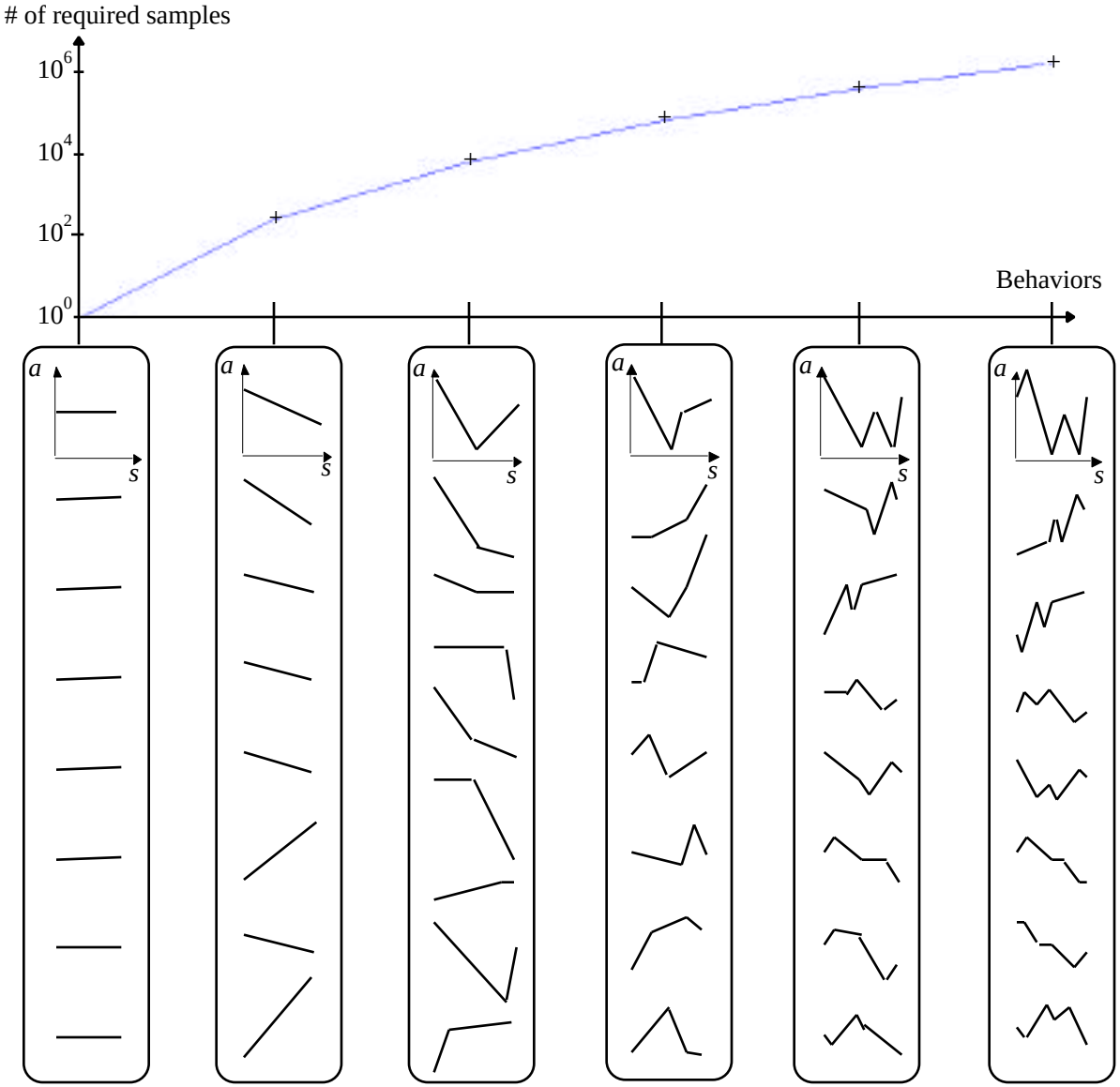


Fig. 46. Illustration of the relation between the "complexity" of a behavior and the number of required learning samples. We have represented below the X-axis a few representative behaviors, starting with the simplest. A behavior involves 8 sensors, therefore we draw 8 sensor-action graphs per behavior. A simpler behavior requires fewer samples to be learned than a more complex one. Note that the Y-axis scale is logarithmic.

The problem is to have some sort of external supervisor, capable of ranking situations by difficulty and of choosing tasks of increasing difficulty [Dorigo et al., 1998]. This process is dependent on the learned policy, requires human supervision and a detailed knowledge of the behavior solution and associated intermediate steps (behaviors of less complexity). Such knowledge needs to be so precise, that one can wonder why, knowing so much, the human operator is not able to directly define the robot policy algorithm? Also, these biases, by carving the path to the behavior solution, do not allow the learning process to find an unexpected solution. Tuning, in this case, seems a term more appropriate than learning.

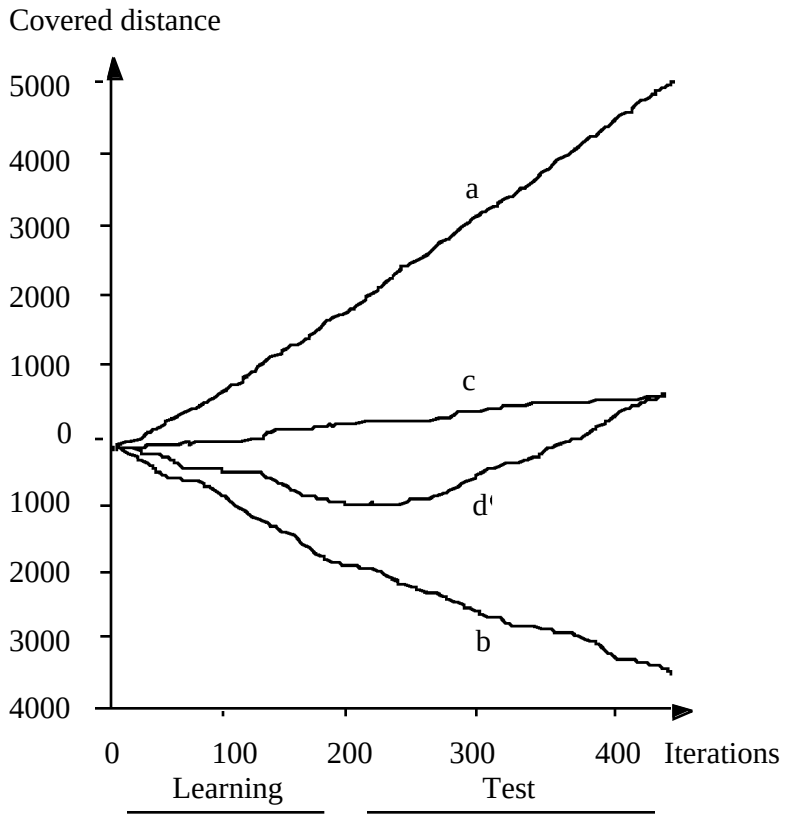
Avoiding *a priori* bias: The main drawback associated with *a priori* biases is that they limit the expressivity of the behavior solution. Only a solution close to the policy described, or allowed, by the biases is allowed. Since the bias validity is, by definition, not guaranteed, there is no guaranty that the final behavior is optimal. *Tabula rasa* learning techniques do not limit the expressivity of the behavior solution, but the search space is so huge that an optimal solution is seldom found. For example, the reinforcement learning algorithm task is to improve the cumulative reward over time, and despite a good learning phase (i.e., no negative rewards experienced during the phase), it often happens that the obtained behavior does not exhibit the expected behavior. Addition of knowledge (*a posteriori* knowledge) can be used to complete or correct the learning process. It allows one to take into account the non-optimal obtained behavior --nevertheless the result of a learning process in a huge search space. *A posteriori* biases can take the form of the addition of external modules dealing with sequences of actions.

Obstacle avoidance behavior correction

RF: +1 if it is avoiding, or

-1 if a collision occurs, or

0 otherwise.



The robot is avoiding when the present sum of sensor values is smaller than the last one, the difference being greater than 0.06. A collision occurs when the sum of the six front sensor values is greater than 2.90, or the sum of the two back sensor values is greater than 1.95.

Fig. 47. Distances covered by Khepera during four different experiments of learning an obstacle avoidance behavior. Behavior (a) displays a predilection for forward moving, (b) prefers backward moving, (c) prefers small forward movements, (d) changes its policy at the end of the learning phase (200 iterations).

The learning can be corrected and improved with the use of a set of forbidden sequences of actions. Here are three sequences of actions to forbid in order to correct the learned behaviors:

1/ Moving back and forth i.e., alternate sequences of actions having the same absolute values.

2/ Small movements i.e., long sequences of actions having small absolute values.

3/ Backward avoidance i.e., long sequences of actions with negative speed values for both motors.

All these sequences modify the exploitation. The effect is to suppress the eligibility of actions in a given situation (and a given historical context). On the self-organizing map implementation, the second closest neuron is selected instead of the first. As shown Fig. 48, using this set of forbidden sequences of actions, only forward behaviors are learned.

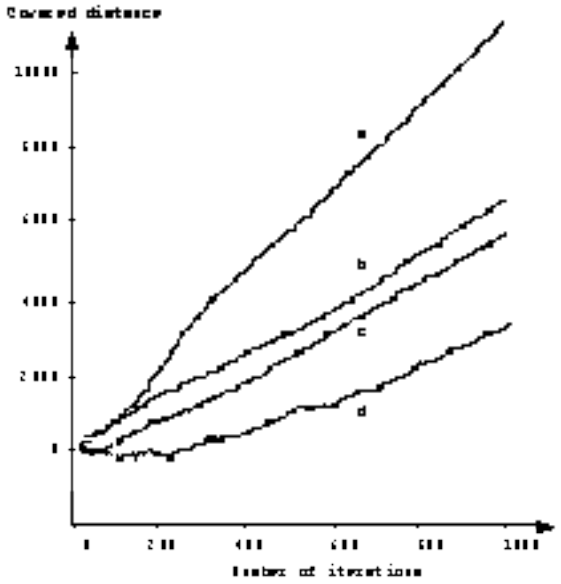


Fig. 48. Covered distances by the four different policies (a, b, c, d) of Fig. 47 after adding the forbidden sequence of actions module and re-starting the learning.

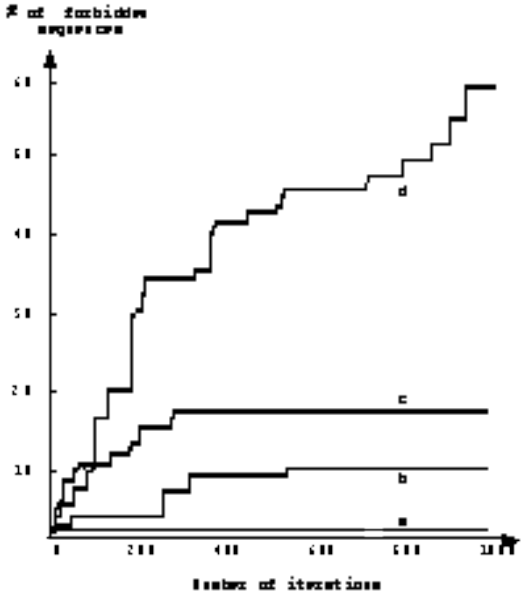


Fig. 49. Number of forbidden sequences used per experiment for the experiences described in fig. 47 and fig. 48.

Only forward behaviors are learned, but there is still a large variability among the proposed solutions. It is interesting to relate the number of times the forbidden sequence module is used with the quality of the obtained behavior. Fig. 49 shows the number of times the *a posteriori* biases are invoked.

Reinforcement Function Design

RFs guide the explration process. In this way, they can be considered as biases. The RF quality is intrinsically limited by the expert'abilities. When a reinforcement learning experiment does not converge, it is impossible to know if it is due to the fact that the experiment was too short and more examples are needed, or if the intrinsic nature of the RF forbids convergence. Today, RL researchers use a slow-and-painful trial and errors approach to define the RF. In the meantime efforts have been devoted to find ways to automatically learn the biases. For example, UPA [Santos 99].

$$RF(s_1,...,s_u) = \begin{cases} +1 \; if \; g_1(s_1,...,s_u) > \theta_+ \\ -1 \; if \; g_2(s_1,...,s_u) < \theta_- \\ 0 \quad otherwise \end{cases}$$

where $(s_1,..., s_u)$ is the output readings of the sensors, $g_1( )$ and $g_2( )$ are any functions linking the sensor data to the rewards.

UPA has been developed to adjust automatically the threshold values: θ_+ and θ_-, optimizing in this case, the exploration part of the learning phase by achieving and maintaining pre-defined ratios of positive and negative rewards. If there is no positive reward, the evaluation function built during the learning phase will have "0" as maximum value and the policy cannot select effective actions. If there is no negative reward, the robot can remain in a dead-end situation forever. If there is no null reward, the evaluation function will be non-continuous at the frontier between positive and negative situation-action pairs.

$$RF((s_1,...,s_{16})^t,(s_1,...,s_{16})^{t-1}) = \begin{cases} +1 \text{ if } g_1((s_1,...,s_{16})^t,(s_1,...,s_{16})^{t-1}) > \theta_+ \\ -1 \text{ if } g_2(s_1,...,s_{16}) < \theta_- \\ 0 \quad \text{otherwise} \end{cases}$$

where $$g_1((s_1,...,s_{16})^t,(s_1,...,s_{16})^{t-1}) = (g_2(s_1,...,s_{16})^t + \sum_{i=7}^{10} s_i^t) - (g_2(s_1,...,s_{16})^{t-1} + \sum_{i=7}^{10} s_i^{t-1})$$

$$g_2(s_1,...,s_{16})^t = \sum_{i=1}^{4} s_i^t + \sum_{i=13}^{16} s_i^t$$

If the update of the threshold values is done continuously during the learning phase, so as to take into account the improvement of the robot policy due to tighter threshold values, then this will in return update the target goal, which will impact the reward ratio, and so forth. This way of modifying and increasing the quality of the target behavior is independent on the learned policy, and therefore does not require some sort of external supervisor, capable of ranking situations by difficulty and of choosing tasks of increasing difficulty as for sample requirement reduction biases.

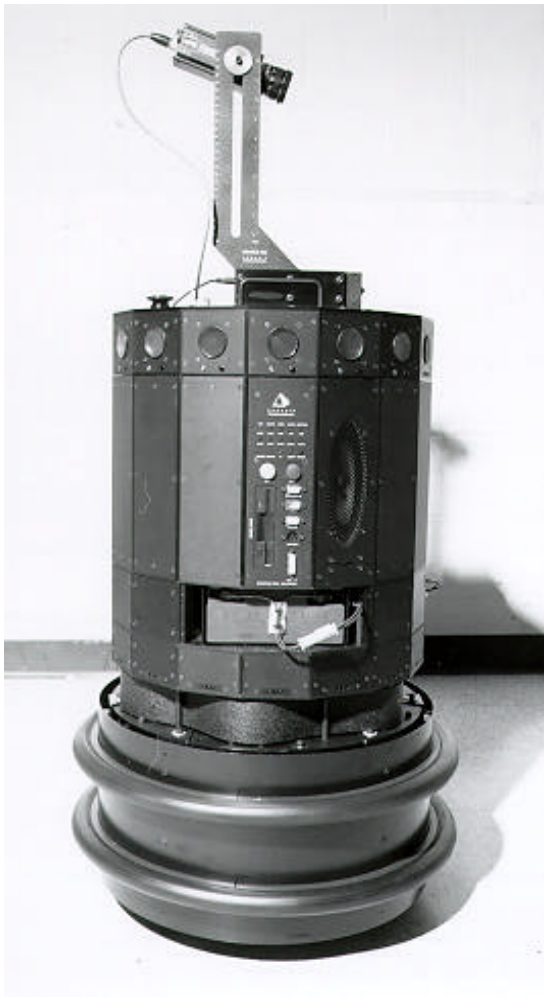


Fig. 50. NOMAD 200 with a ring of 16 IR sensors and 16 SONAR sensors.

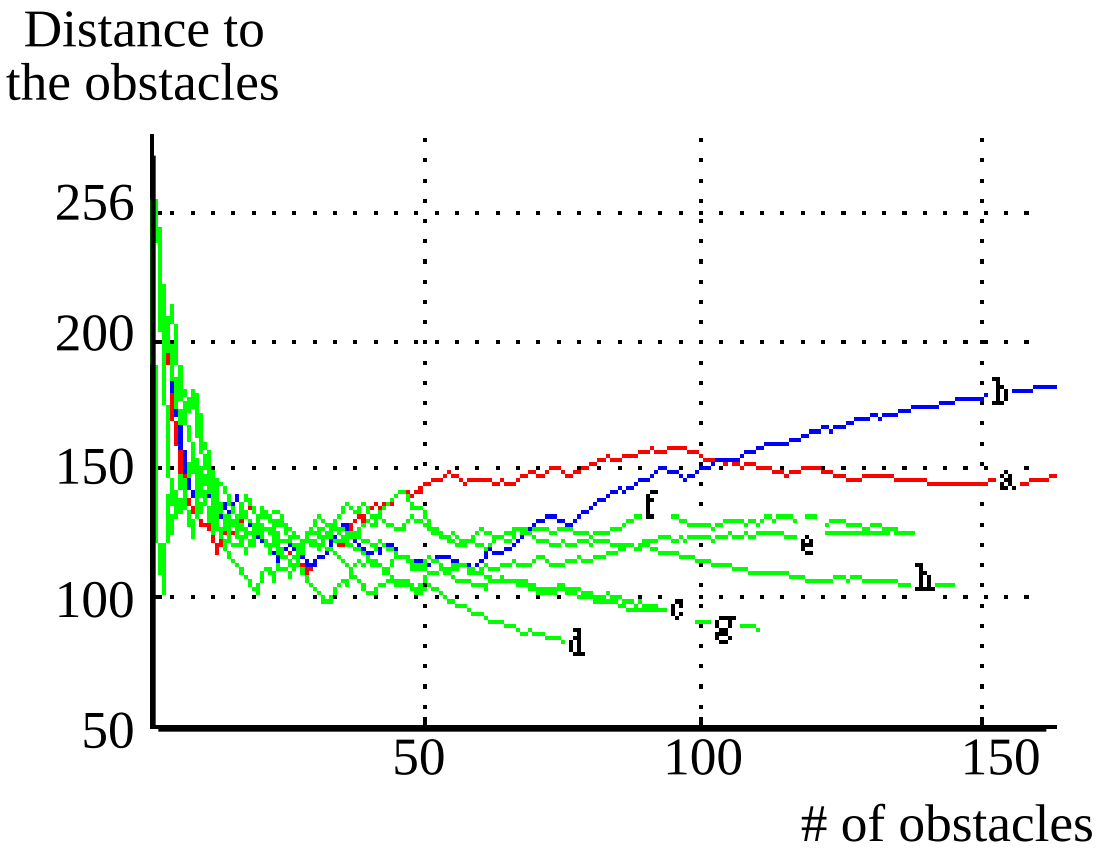


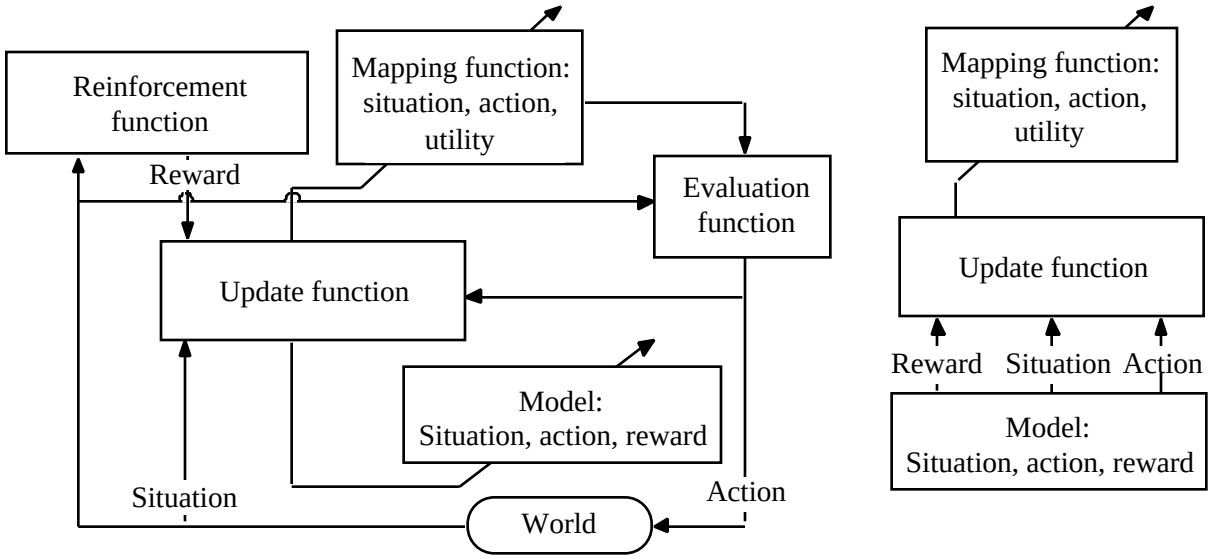
Fig. 51. Distance to the obstacles with respect to the number of obstacles encountered. There are 300 iterations (200 learning plus 100 test). The total number of encountered obstacles provides an indication of the efficiency of the robot policy: the greater the efficiency the more numerous the encountered obstacles.

(a) is obtained by a random move selection behavior (no learning is involved),
(b) is the learned behavior using the values given by UPA ($\theta_+ = 390$, $\theta_- = 1150$),
(c) ($\theta_+ = 195$, $\theta_- = 2300$), (d) ($\theta_+ = 780$, $\theta_- = 575$),
(e) ($\theta_+ = 195$, $\theta_- = 1150$), (f) ($\theta_+ = 780$, $\theta_- = 1150$),
(g) ($\theta_+ = 390$, $\theta_- = 2300$), (h) ($\theta_+ = 390$, $\theta_- = 575$).

Building of a non-explicit Model

A priori biases intend to reduce the number of samples required to achieve the learning, because only a limited number of moves can be made during a robotic experiment (see fig. 3). *A priori* biases could be avoided if more samples can be used for the learning. The usual idea for adding samples is to use a model to generate synthetic samples. We have discarded this option (see introduction) because of the large involvement required from the user -- antinomic to the automatic aspect associated with learning. But, there are non-explicit models that do not require user involvement. Dyna is one of the first attempts to use non-explicit models to speed-up the learning (in fact the temporal credit assignment); lazy learning is another attempt.

DYNA: Without building an explicit model, the DYNA architecture [Sutton, 1991] re-run previously seen (in the real world) situation-action pairs so as to back propagate delayed rewards. The returned reward is the same as in the real world. When the experience is performed in the real world, the exploration function is the maximum function. Otherwise, the exploration function is a random function which leads to non-zero reinforcement rewards only in previously seen (in the real world) situation-action pairs. Because the world model is not explicit, it is called an *internal* world model (Fig. 52). For each real experience with the world, many hypothetical experiences randomly generated can also be processed and learned from. The cumulative effect of these synthetic experiences is that the policy approaches the optimal policy given by the current samples.



a b

Fig. 52. Dyna architecture: for each Q-learning iteration in the real world (a), many "hypothetical" iterations are conducted (b) using the internal world model so as to backpropagate delayed reward information.

Prioritized Sweeping [Moore et al. 1993] and Queue-Dyna [Peng and al., 1993] improve on Dyna by concentrating on "interesting" parts of the search space, instead of randomly selecting situation-action pairs. A updating priority is added to the situation. This priority depends on the Q-value change size, favoring "new-unforeseen-important" transitions and the backpropagation of the associated rewards.

Lazy RL

Dyna helps to converge to the solution if it is contained in the samples, but the problem is to obtain a set of samples large enough to allow the learning of a behavior solution. New additional samples are required. *Lazy learning* [Aha, 1997], also called instance-based learning, provides a way to achieve this result. In a lazy learning approach (Fig. 53), the computation of the inputs is delayed until the necessity arises. Lazy learning samples the situation-action space, storing the succession of events in memory and, when needed, probes the associative memory for the best move. The sampling process stores the successive situation-action pairs generated by a random action selection policy. The exploration phase is done only once, stored and used later by all future experiments. The probing of the memory involves complicated computations: clustering, pattern matching, and so forth.

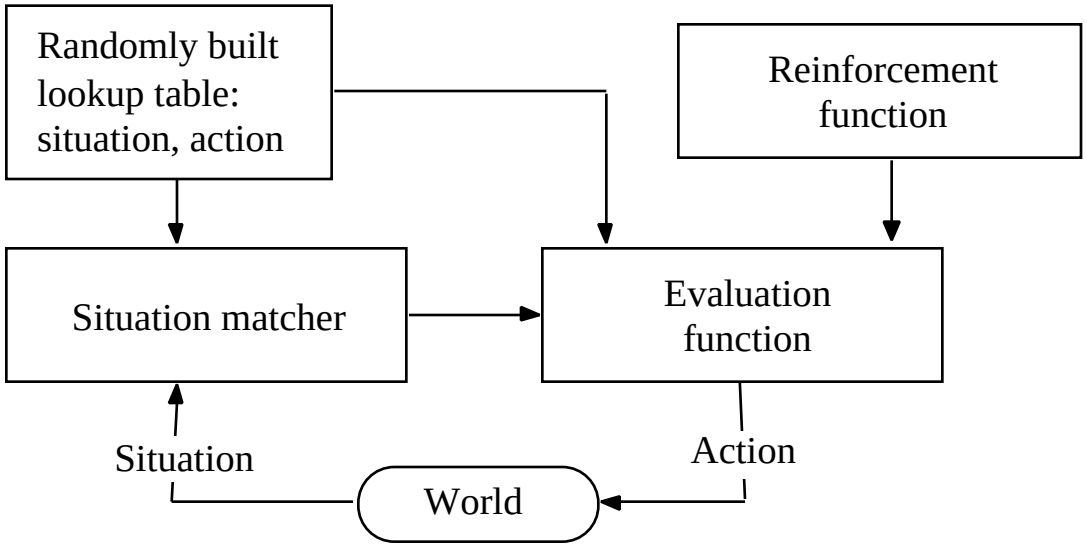


Fig. 53. Lazy learning used as initial knowledge. Randomly sampled situation-action pairs in the lookup table are used by the situation matcher to determine the effect of actions. Possible incoming situations are evaluated and ranked by the evaluation function with the help of the reinforcement function. The best rewarding action is conducted.

By storing situation-action pairs, a lazy memory builds a model of the situation transition function. Two questions immediately arise about the legitimacy of considering lazy learning as a model, and if so, about the quality of the model:

- Lazy learning assumes that the environment is not changing. Any change in the environment diminishes the quality of the bias provided by the lazy memory. But, basic features, such the effects of moving forward or backward in front of an obstacle tend to persist despite environment variations and are nevertheless important usable knowledge.

- The lazy memory (or model) is used as a bias to leverage the model-free following learning phase (Q-learning). It has been demonstrated [Whitehead, 1991] that random exploration might be dangerous and in some environments is an immensely ineffective method of gathering data, requiring exponentially more data than a system that interleaves experience gathering with policy-building more tightly. However, these results only apply to the “go to a particular location” type of applications and do not generalize to more “reactive” behaviors like obstacle avoidance or target observation. Therefore, we assume that the model is correct in the context of providing a bias to leverage a model-free learning phase that follows.

Using the lazy memory, a number of algorithms can be applied to find a good policy, like value iteration, policy iteration or a mix of both. However, since there is no question that we are here only looking for a bias, that a Q-learning phase will also take place, then the quality of the resulting bias policy is not as important as the computational cost. Sheppard et al. [Sheppard et al., 1997] propose to mix lazy learning and reinforcement learning, probing the memory with the RF. Their objective is to provide a method for predicting the rewards for some state-action pairs without explicitly generating them. They call their algorithm *lazy Q-learning*. For the current real world situation, a situation matcher locates all the states in the memory that are within a given distance. If the situation matcher has failed to find any nearby situations, the action comparator selects an action at random. Otherwise, the action comparator examines the expected rewards associated with each of these situations and selects the action with the highest expected reward. This action is then executed, resulting in a new situation. There is a fixed probability (0.3) of generating a random action regardless of the outcome of the situation matcher. New situation-action pairs are added to the memory, along with a Q-value computed in the classical way. Among similar situation-action pairs in the memory, an update of the stored Q-values is made. There is a limit to the genericity of this lazy memory because the Q-values associated with the situation-action pairs only apply for a particular application.

Cooperative multi-robot observation of multiple moving targets (or CMOMMT for short) application [Parker, 1997]. In a bounded arena (Fig. 54), a team of robots with 360˚ field of view of limited range has to maximize the observation time of a set of targets moving randomly (5% probability of change of direction, maximum speed less than the maximum robot speed). We say that a robot is monitoring a target when the target is within the robot's sensory field of view. The objective is to maximize the collective time during which targets are being monitored by at least one robot. The radius of the sensory robot range is less than the size of the arena, implying that robots have to move to maintain observational contact.

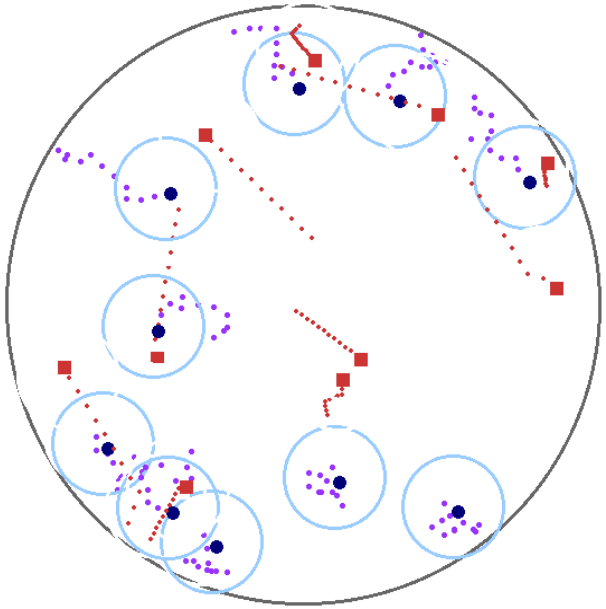


Fig. 54. Bounded arena, with 10 robots using the lazy Q-learned behavior (no awareness involved). The radius of the arena is 5, the radius of the sensory perception range of the robots is 1. There are 10 randomly moving targets. The dotted lines indicate the paths followed by the robots and the targets. The targets have different speeds: the closer the dots, the smaller the speed.

Predictions in CMOMMT

The lazy memory is built using a random action selection policy for the robots and recording at each time step the total number of targets under observation by the team. It is important to be able to ascertain the quality of the lazy memory – in fact the quality of the non-explicit model that has been build. Certainly, the larger the number of samples in the memory, the better the performance we can expect from the following Q-learning phase. However, we need to know before the Q-learning phase starts, that the memory will prove useful. The coherence of the memory can be measured and compared to a incoherent memory. The coherence of the memory is demonstrated by the consistency with which positive rewards lead to positive reward in similar situations, null rewards lead to null rewards in similar situations and negative rewards lead to negative rewards in similar situations. The larger the number of similar situations the better the quality of the demonstration. It is important to note that the maximum number of situations that can be considered as similar is directly proportional to the size of the memory (the larger the better).

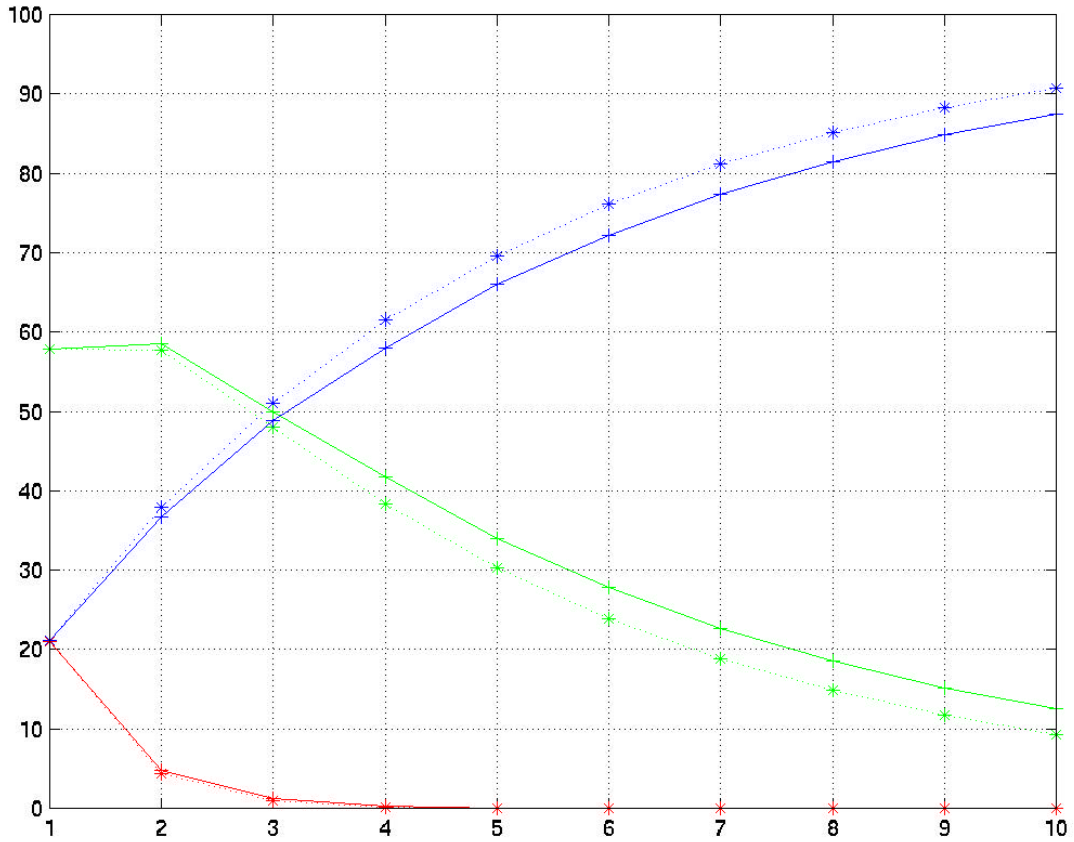
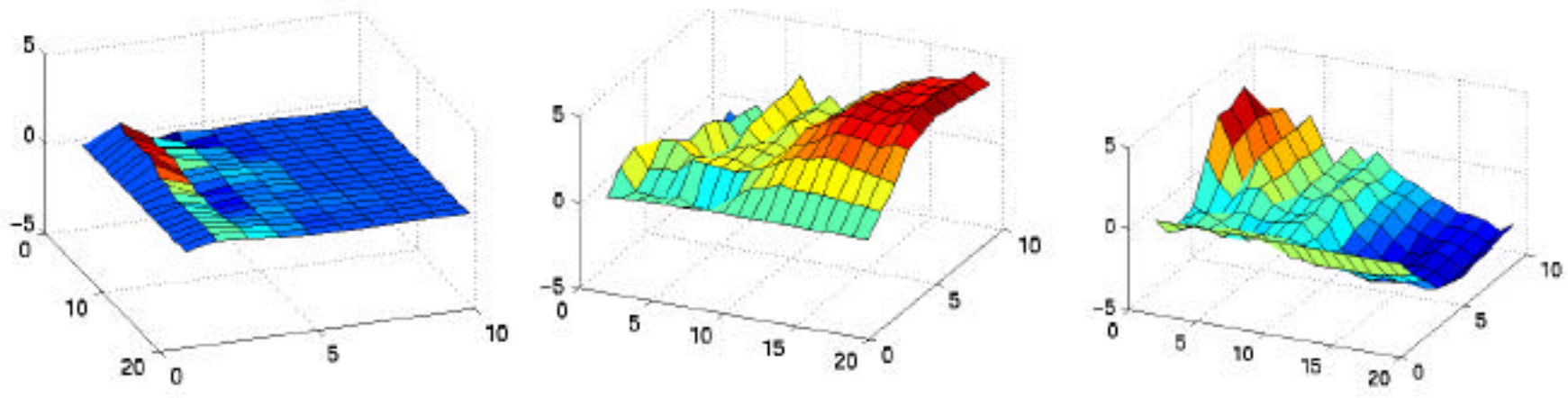


Fig. 55. Coherence of the non-explicit model build by the memory. In dotted lines, the incoherent memory; in plain lines the actual measures. The differences between incoherent memory and the actual one suggest that negative situation-action regions are continuous as are null-reward regions. There is no coherence for the positive rewards, which means there is no (represented) region in the memory that is rewarding.

Fig. 56 presents the differences (Z-axis) in percentages of positive, null and negative rewards (measured values - incoherent memory), with respect to the size of the memory and the size of the “similar” situation sets. As we can see, gain is directly proportional to the size of the memory and to the size of the set of similar situations. Also, a large memory tends to smooth the surface (by reducing the standard deviation).



a b c

Fig. 56. Differences between the actual memory and its incoherent version.

(a) Positive rewards are very similar – impossible to predict.

(b) Null rewards are more numerous, and

(c) Negative rewards are less numerous: there are specific regions of the search space that encode null rewards and others that encodes negative rewards.

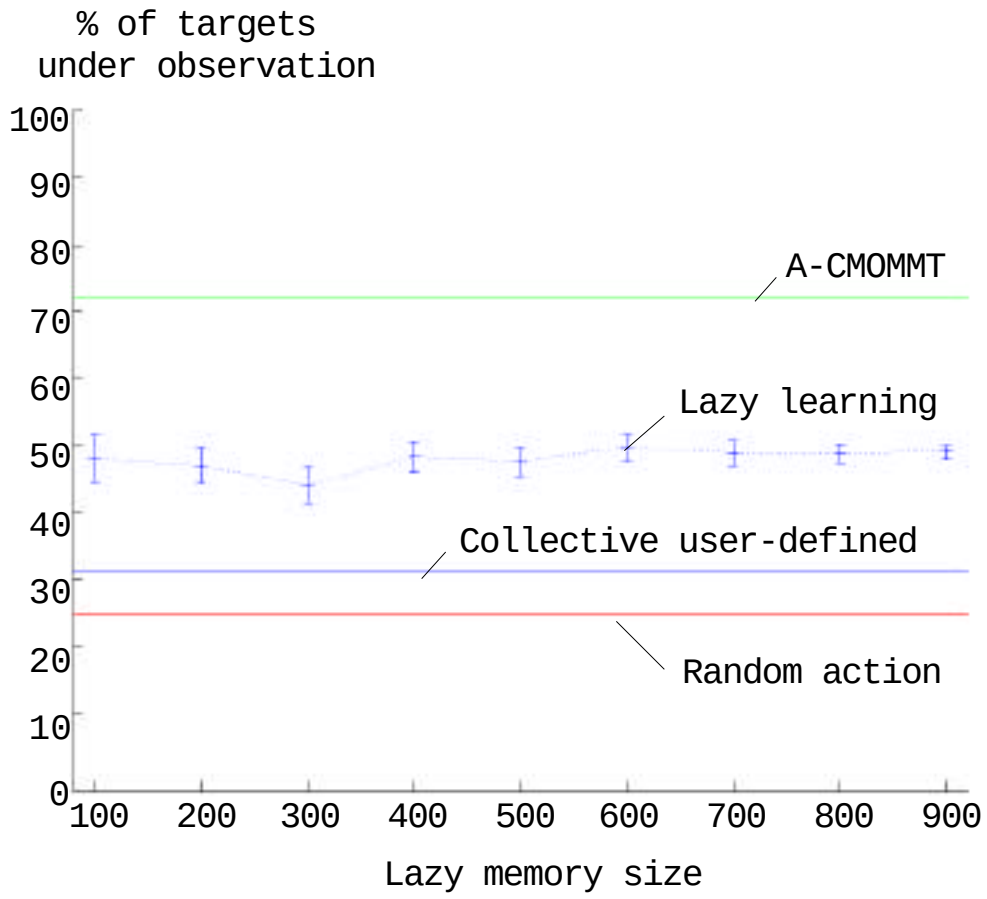


Fig. 57. Performances of the non cooperative lazy Q-learning compared to a random action selection policy, a user-defined non cooperative policy and A-CMOMMT. The size of the lazy memory varies between 100 to 900 situation-action pairs. There are 10 robots and 10 randomly moving targets. The results are the mean of 10 different experiments per point for lazy learning policy, and 100 experiments for the other 3 policies. Each experiment duration is 1000 iterations.

References

Aha D. (ed.), *Lazy Learning*, Kluwer, 1997.

Anderson J. A., *An introdution to neural networks*, Bradford - MIT Press, 1995.

Asada M., S. Noda, S. Tawaratsumida and K. Hosoda, “Purposive Behavior Acquisition for a Real Robot by Vision-based Reinforcement Learning,” joint special issue (Guest Eds. H. Hexmoor and M. Mataric) on Learning in Autonomous Robots, *Machine Learning*, 31(1-3), and *Autonomous Robots*, 5(3-4), Jul./Aug. 1998.

Ackley D. & M. Littman, "Interactions Between Learning and Evolution," *Artificial Life II*, SFI Studies Sc. Complexity, vol.X, C. G. Langton & Co Eds. Addison-Wesley, 487-509, 1991.

Barto A. G. and Anandan P., “Pattern Recognizing Stochastic Learning Automata,” *IEEE Trans. on Systems, Man and Cybernetics*, SMC-15 : 360-375, 1985.

Baxter J., “Theoretical Models of Learning to Learn,” in *Learning to Learn* , S. Thrun and L. Pratt (eds.), Kluwer Academic Publishers, 1998.

Braitenberg V., *Vehicles: Experiments in Synthetic Psychology*, MIT Press, 1984.

Brooks R., "Intelligence without reason," *IJCAI'91*, Sydney, 1991.

Brooks R., C. Breazeal (Ferrell), R. Irie, C. C. Kemp, M. Marjanovic, B. Scassellati, M. Williamson, “Alternate Essences of Intelligence”, *Proc. of the AAAI 98 Conf.*, 1998.

Cao Y. U., A. Fukuaga and A. Kahng, "Cooperative Mobile Robotics: Antecedent and Directions," *Autonomous Robots* 4, 7-27, 1997.

Colombetti M., M. Dorigo and G. Borghi, "Behavior Analysis and Training - A Methodology for Behavior Engineering," Special Issue on Learning Autonomous Robots, M. Dorigo Guest Editor, IEEE SMC-part B, Vol. 26, No. 3, 365-380, June 1996.

Crites R. and A. Barto, “Improving elevator performance using reinforcement learning,” in D. S. Touretzky, M. C. Mozer, and M. E. Hasselmo (eds), *Advances in Neural Information Processing Systems: Proceedings of the 1995 Conference*, pp. 1017-1023, MIT Press, Cambridge, MA, 1996.

Darrell T., “Reinforcement Learning of Active Recognition Behaviors,” Interval Research Technical Report 1997-045. (http://www.interval.com/papers/1997-045 - Portions of this paper previously appeared in Advances in Neural Information Processing Systems 8, (NIPS '95), pp. 858-864, MIT Press, and Intelligent Robotic Systems, M. Vidyasagar ed., pp. 73-80, Tata Press, 1998.

Dayan P. and T. Sejnowski, "TD(l) convergences with probability 1," *Machine Learning*, 14(3), 1994.

Dorigo, M., "Introduction to the Special Issue on Learning Autonomous Robots," M. Dorigo (guest editor), IEEE Trans. on Systems, Man and Cybernetics - part B, Vol. 26, No. 3, 361-364, 1996.

Dorigo M. and H. Bersini, “A comparison of Q-learning and classifier systems,” *Proc. of the 3rd Int. Conf. on Simulation of Adaptive Behavior* (SAB'94), Brighton, UK, MIT Press, August 1994.

Dorigo M. & M. Colombetti, "Training Agents to Perform Sequential Behavior," *Adaptive Behavior*, MIT Press, 2 (3), pp. 247-276, 1994.

Dorigo M. and M. Colombetti, *Robot Shaping: An Experiment in Behavior Engineering*, MIT Press, 1998.

Heemskerk, J. and N. Sharkey, "Learning Subsumptions for an Autonomous Robot," IEE seminar on self-learning robot, Digest No: 96/026, Savoy Place London, 12, England, February 1996.

John H. Holland (1975), *Adaptation in Natural and Artificial System*, reprinted by MIT Press in 1992.

Hexmoor H., L. Meeden, and R. Murphy (1997), “Is Robot Learning a New Subfield?”, *AI magazine*, 1997.

Kaelbling, L., Littman, M., and A. Moore, "Reinforcement Learning: A Survey," Journal of Artificial Intelligence Research 4, 237-285, 1996.

Kalmar Z., C. Szepesvari and A. Lorincz, “Experiments with a Real Robot,” joint special issue (Guest Eds. H. Hexmoor and M. Mataric) on Learning in Autonomous Robots, *Machine Learning*, 31(1-3), and *Autonomous Robots*, 5(3-4), Jul./Aug. 1998.

Kohonen T., *Self-Organisation and Associative Memory*, Springer-Verlag, Berlin, 1984.

Kretchmar R. M. and C. W. Anderson, “Comparison of CMACs and Radial Basis Functions for Local Function Approximators in Reinforcement Learning,” *Proc. of ICNN'97*, Houston, Texas, USA, June 1997.

Lin L-J., "Reinforcement Learning for Robots Using Neural Networks," Ph.D. thesis, Carnegie Mellon University, Pittsburgh, CMU-CS-93-103, January 1993.

Lin L. J., "Self-improving reactive agents based on reinforcement learning, planning and teaching," *Machine Learning* 8: 293-321, 1992.

Littman, M. L., “Memoryless policies: theoretical limitations and practical results,” *From Animals to Animats 3: Proc. of the Third Int. Conf. on Simulation of Adaptive Behavior*, Cliff, d., Husbands, P. Meyer, J-A. and Wilson, S. W. (Eds.), Cambridge, MA, MIT Press, 1996.

McCallum, R. A., “Instance-based State Identification for Reinforcement Learning,” in Advances In Neural Information Processing Systems 7, MIT Press, 1995.

Maclin R. and J. W. Shavlik, “Creating Advice-Taking Reinforcement Learners,” in *Learning to Learn* , S. Thrun and L. Pratt (eds.), Kluwer Academic Publishers, 1998.

Mahadevan S. and J. Connell, "Automatic Programming of Behavior-based Robots using Reinforcement Learning," *Artificial Intelligence*, 55, 2, pp. 311-365, July 1991.

Mahadevan S., “Rapid Concept Learning for Mobile Robots,” joint special issue (Guest Eds. H. Hexmoor and M. Mataric) on Learning in Autonomous Robots, *Machine Learning*, 31(1-3), and *Autonomous Robots*, 5(3-4), Jul./Aug. 1998.

Mataric M. J., “Reinforcement Learning in Multi-Robot Domain,” *Autonomous Robots* 4, 73-83, 1997.

Michaud, F. and M. J. Mataric, “Learning from History for Behavior-Based Mobile Robots in Non-Stationary Conditions,” Special joint issue Machine Learning and Autonomous Robots Journals, H. Hexmoor and M. Mataric (guests eds.), 1998.

Millán J. del R., "Rapid, Safe and Incremental Learning of Navigation Strategies," Special Issue on Learning Autonomous Robots, M. Dorigo Guest Editor, *IEEE Trans. on Systems, Man and Cybernetics - part B*, Vol. 26, No. 3, 408-420, June 1996.

Mitchell T., *Machine Learning*, McGraw Hill, March 1997 (pp. 107).

Mondada F., E. Franzi & P. Ienne, "Mobile Robot Miniaturisation: A Tool for Investigation in Control Algorithms," *Third International Symposium on Experimental Robotics*, Kyoto, Japan, October 1993.

Moore A. W. and C. G. Atkeson, Prioritized sweeping: Reinforcement learning with less data and less real time, *Machine Learning*, 13, 1993.

Parker L. E., "Cooperative Motion Control for Multi-Target Observation," *Proc. IROS 97*, Grenoble, France, 1997.

Peng J. and R. J. Williams, Efficient learning and planning within the Dyna framework, *Adaptive Behavior*, 1(4), 437-454, 1993.

Rumelhart D. , G. Hinton & R. Williams, "Learning internal representations by error propagation," *Parallel Distributed Processing*, Vol. 1, D. Rumelhart & J. McClelland Eds. Cambridge, MIT Press, pp. 318-362, 1986.

Santos J. M. and C. Touzet, “Exploration Tuned Reinforcement Function,” to appear in *Neurocomputing*, 1999.

Sharkey N., “Learning from Innate Behaviors: A Quantitative Evaluation of Neural Network Controllers,” joint special issue (Guest Eds. H. Hexmoor and M. Mataric) on Learning in Autonomous Robots, *Machine Learning*, 31(1-3), and *Autonomous Robots*, 5(3-4), Jul./Aug. 1998.

Sheppard J. W. and S. L. Salzberg, "A Teaching Strategy for Memory-Based Control," Lazy Learning, D. Aha (ed.), Kluwer Academic Publishers, 343-370, 1997.

Schmidhuber J. and J. Zhao, “Multi-Agent Learning with the Success-Story Algorithm,” Distributed Artificial Intelligence Meets Machine Learning – Learning in Multi-Agent Environments, G. Weiss (ed.), Lecture Notes in Artificial Intelligence, Vol. 1221, Springer-Verlag, 82-93, 1997.

Singh, S. P., T. Jaakkola and M. Jordan, “Learning Without State-Estimation in Partially Observable Markovian Decision Processes,” Proc. of the Eleventh Int. Machine Learning Conf., 1994.

Sutton R. and A. Barto, *Reinforcement Learning*, Bradford Book, 1998.

Sutton R. S., “Reinforcement Learning Architectures for Animats,” *Proc. of the First Int. Conf. on Simulation of Adaptive Behavior, From Animals to Animats*, Edited by J-A Meyer and S. W. Wilson, MIT Press, 288-296, 1991.

Tesauro, G., “Temporal difference learning and TD-Gammon,” *Communications of the ACM*, 38:58-68, 1995.

Thrun S., “Efficient Exploration In Reinforcement Learning,” TR CMU-CS-92-102, January 1992.

Thrun S., “Exploration and Model Building in Mobile Robot Domains,” *Proceedings of IEEE International Conference on Neural Networks*, IEEE Neural Networks Council, 1993.

Thrun S. and L. Pratt, “Introduction and Overview,” in *Learning to Learn* , S. Thrun and L. Pratt (eds.), Kluwer Academic Publishers, 1998.

Touzet C., “Neural Reinforcement Learning for Behaviour Synthesis,” Special issue on Learning Robot: the New Wave, N. Sharkey (guest ed.), *Robotics and Autonomous Systems 22*, No 3-4, 251-281, 1997.

Touzet C., Sehad S. and Giambiasi N., “Improving Reinforcement Learning of Obstacle Avoidance Behavior with Forbidden Sequences of Actions,” *International Conference on Robotics and Manufacturing*, Cancun, Mexico, 14-16 June 1995.

Watkins C. J. C. H., *Learning from Delayed Rewards*, Ph.D. thesis, King's College, Cambridge, England, 1989.

Watkins C. and P. Dayan, “Q-Learning,” *Machine Learning*, 8:279-292, 1992.

Whitehead S., J. Karlsson and J. Tenenberg, "Learning Multiple Goal Behavior via Task Decomposition and Dynamic Policy Merging," Robot Learning, J. Connell and S. Mahadevan (eds.), Kluwer Academic Publishers, 1993.

Whitehead S. D, “A Complexity Analysis of Cooperative Mechanisms in Reinforcement Learning,” *Proc. of AAAI*, Vol. 2, pp. 607-613, 1991.

Some starting points to explore the web on the tracks of agent learning (July 1999)

Adaptive Behavior (Society & Journal)
http://adaptive-behavior.org/

Home Pages of Machine Learning & CBR Folks
http://www.aic.nrl.navy.mil/~aha/people.html

Adaptive Intelligent Systems Architecture
http://ai.eecs.umich.edu/cogarch0/ais/index.html

lectures.html
http://www.csee.usf.edu/~mahadeva/ml-class/lectures.html

The Reinforcement Learning Group at Carnegie Mellon
http://www.cs.cmu.edu/afs/cs.cmu.edu/project/reinforcement/mosaic/homepage.html

MIT Artificial Intelligence Laboratory Home Page
http://www.ai.mit.edu/

Khepera
http://www.k-team.com/robots/khepera/

Claude Touzet
http://saturn.epm.ornl.gov/~touzetc/