HIERARCHICAL REINFORCEMENT LEARNING AND DECISION MAKING FOR INTELLIGENT MACHINES

PEDRO LIMA, GEORGE SARIDIS
Electrical, Computer and Systems Engineering Department
Rensselaer Polytechnic Institute
Troy, NY 12180-3590

Abstract

A methodology for performance improvement of Intelligent Machines based on Hierarchical Reinforcement Learning is introduced. Machine Decision Making and Learning are based on a cost function which includes reliability and computational cost of algorithms at the three levels of the hierarchy proposed by Saridis. Despite this particular formalization, the methodology intends to be sufficiently general to encompass different types of architectures and applications.

Novel contributions of this work include the definition of a cost function combining reliability *and* complexity*, recursively improved through feedback, a Hierarchical Reinforcement Learning and Decision Making algorithm which uses that cost function, and a methodology supported on Information-Based Complexity for joint measure of algorithm cost and reliability.*

Results of simulations show the application of the formalism to Intelligent Robotic Systems.

1 Introduction

Most of the work done in the last few years towards building Hierarchical Intelligent Machines (IM) [8] quite often mentions the need for a methodology of designing the IM and a measure of how successful the final result is. An analytic design based on measures of performance recursively improved through feedback, assures some degree of certainty about the measurability and reliability of that design.

The methodology intends to be sufficiently general to encompass different types of architectures and applications. Despite this, the work described is developed within the framework of the Analytic Theory of Intelligent Machines developed by Saridis et al [9]. Previous results within this framework established a general architecture for the IM and detailed this architecture for the different levels. However, the flow of feedback through the hierarchy with the purpose of improving the overall performance by updating the decision making structure, has never been detailed for the complete hierarchy. Furthermore, even though the general goal is to decrease *entropy* at all levels, and reliability has been proposed as an equivalent measure of entropy before[5], neither cost has ever been included in the cost function, nor has a recursive estimate of reliability been considered.

The present work proposes a methodology for performance improvement of Intelligent Machines based on *Hierarchical Reinforcement Learning*. Different options to accomplish a goal or a subgoal may be found at all levels of the IM: the **Organization Level** has to decide among different *tasks* capable of executing a given goal (*command*) sent to the machine; given the chosen task, composed by subgoals (*events*), the **Coordination Level** has to determine, for each event, the best among the set of *primitive algorithms* capable of solving each subgoal. A cost function is necessary to compare the different alternatives at each level. The proposed procedure recursively estimates a cost function combining *reliability* and *computational cost* of tasks, events and primitive algorithms.

This approach has the advantage of providing a cost measure applicable to several different problems, since it is based on *reliability* (defined as the probability that an algorithm will meet some set of specifications in a given state of the environment) and *complexity* of a problem, i.e. minimum computational cost of the algorithm which solves the problem, here considered not only in terms of computation time (time-complexity) but also more general features such as memory and other resources usage (space-complexity). These are sufficiently general measures in the sense that the success of any primitive algorithm (e.g. a controller, a sensor), event or task, can be measured by determining how reliable the algorithm is, while imposing some complexity constraint(s).

The paper is organized as follows: after this introduction, section 2 defines the hierarchical, goal-oriented architecture assumed in the sequel and explains the corresponding decision making and learning algorithms, including the definition of the cost function. Afterwards, section 3 describes the application to an Intelligent Robotic System, and section 4 presents preliminary conclusions and directions of future work.

2 Hierarchical Reinforcement Learning and Decision Making

When dealing with large complex systems of different types, such as industrial processes or mobile robots, common questions are "how to measure performance?" and "how to improve the performance measure?". Usual performance measures are energy consumption and final product quality for process industries, or whether the mission was accomplished or not for an autonomous mobile robot. However, these are measures pertaining to each system. Furthermore, it is very difficult, if not impossible, to relate these performance variations to the performance of the underlying subsystems, in order to learn how to improve performance.

Here, we assume that each subsystem is designed to achieve its best possible performance, in the sense of not failing to meet its specifications the maximum possible number of times, without using too many resources. Assuming a fixed cost, in terms of resources consumption, if we can monitor whether a subsystem fails to meet its specifications each time its service is required, then it is possible to learn along time the best subsystem among a set of pre-designed alternatives, for each subgoal. Overall, the best task which accomplishes the main goal, is chosen based on the performance of the subsystems composing the task. The approach provides a measure to compare different designs, which will be distinguished by the quality of the pre-designed algorithms and the way they are composed into tasks. It also provides the methodology to obtain convergence to the best possible solution given a design. Better designs will converge to smaller cost functions. Furthermore, it provides a simple way of improving performance through feedback, here consisting of success/failure signals only.

The design process follows a *bottom-up* approach, where the alternative primitive algorithms capable of implementing the problem represented by an event and the different events feasible for a given command are pre-specified. Subsequently, the *planning* problem is to **compose** these events to form a task, as opposed to a *top-down* approach, where an original goal is recursively decomposed until a feasible task is obtained.

On the other hand, the execution process flows both top-down (for decision-making) and bottom-up (for feedback). For each command, a task is selected based on the current probability distribution over the set of tasks. A task, in turn, is a sequence of events. The primitive algorithms translating each event are selected based on the current probability distribution over the set of algorithms for the event. Feedback from the environment where the IM operates consists of successes and failures of the algorithms to meet their specifications. These are used to update in a bottom-up fashion the probability distributions at the decision levels.

2.1 The Hierarchical, Goal-Oriented Intelligent Machine

In this subsection we formally define the components of the hierarchical IM.

A *command* is received by the machine from an external subject. This may either be an operator or a subgoal of a mission or plan. However, this command is the goal for the IM and may be translated by more than one *task*. $T^i = \{t_1^i, \ldots, t_{l_i}^i\}$ is the set of *tasks* capable of implementing *command* c_i, $i = 1, \ldots, nc$. A *task* is an ordered sequence of *events*. E is the set of all *primitive events*. $|E| = ne$. $E^i = \{e_1^i, \ldots, {e_{m_i}}^i\} \subset E$ is the set of *primitive events* compatible with *command* c_i. This definition reduces the search space when looking for events to compose a task. We further associate to command i a *language* L_i whose set of *terminal symbols* is E^i. Hence, task t_j^i is a string of L_i. Moreover, $T^i \subset L_i = {E^i}^+$, $|T^i| = l_i$, that is, task t_j^i, $j = 1, \ldots, l_i$ is one of the possible strings composed by elements of E^i, excluding the null string. The size of T^i depends on the constraints imposed by a given command to its compatible primitive events. These constraints are expressed by a *grammar* G_i which generates T^i and whose start symbol represents command c_i. Alternative tasks for a given command have different abilities to achieve the corresponding goal. This is reflected by a probability density function defined over the set of tasks, introduced in the next subsection. Hence, T^i must be a *stochastic language*, and G_i must be a *proper stochastic grammar*[1].

An *event* is represented by a non-null string of primitive events. Notice that events are non-terminal symbols of the grammar G_i which generates T^i. A *primitive event* is an event which is no further decomposable. It represents a problem which can be solved by some *primitive algorithm*. For each event there is at least one algorithm which translates the event, e. g., one *primitive algorithm* which can solve the problem represented by the event. Primitive events are terminal symbols of the grammar G_i which generates T^i. $A^k = \{a_1^k, \ldots, a_{n_k}{}^k\}$ is the set of alternative *primitive algorithms* which may translate *primitive event* e_k, $k = 1, \ldots, ne$.

2.2 Hierarchical Decision Making

There are 2 levels of decision in the hierarchy described in the previous subsection: the Organization Level, where commands are translated into tasks, and the Coordination Level, where primitive events composing a task are translated into primitive algorithms. At each level we need a performance measure to assist the decision making process. To avoid local minima, there must be, at each step, a chance of exploring decisions which are not the current best action. With this purpose, the IM is modeled by a 2-level *Hierarchical Stochastic Automaton* (HSA)[6]. At each level, and at each step, the IM chooses the task or primitive algorithm to apply by random decision, based on the probability distribution function over the corresponding set. The probabilities are updated along time, based on the update of the cost function estimates.

2.3 Cost Function

As explained before, it is desirable that the Intelligent Machine chooses the task and algorithms which **minimize** the Cost while **maximizing** the Reliability. The following cost function is proposed:

$$J_{a\sigma} = 1 - R_{a\sigma} + \rho C_{a\sigma} \tag{1}$$

where R is the reliability, C the cost and ρ a weight factor which balances the two. In general ρ will be such that $\rho C \in [0, 1]$, so that the cost does not overwhelm the reliability when directing the search for the optimal action. The same cost function is used for the 2 levels of decision. Hence, a represents either a task or a primitive algorithm, depending on the level under consideration. The cost function is conditioned by the state of the environment σ, which is assumed to be multi-state and stochastic.

Reliability and Cost of a primitive algorithm in (1) are determined given the desired accuracy (e. g., the error with respect to the specifications for the problem to be solved by the algorithm) and the formalism described by the authors in [3]. Rules for propagating bottom-up the cost function, from algorithms to events and from these to tasks are presented in [4].

2.4 Hierarchical Reinforcement Learning

Until now we have been talking of the actual cost function. Such a cost function may be determined by a model-based approach [3, 5]. The optimal design, in the sense of minimizing (1), is obtained off-line. However, this is impractical in most cases, hence we are interested in improving along time an estimate of the cost function, using feedback from the environment. A recursive estimate also provides some capability of adaptiveness to environment changes, thus reducing the need to predict all environment states. A model-based determination of Reliability assumes a stationary environment.

At the lowest level of the hierarchy, the procedure recursively estimates Reliability of the primitive algorithms. This estimation is subsequently propagated to the upper levels, where it is added to the Cost to obtain the cost function estimate. These estimates of J are used by *Reinforcement Learning Algorithms* to update the *subjective probabilities* of the different algorithms capable of translating a *primitive event* and the different *tasks* capable of translating a command sent to the machine. The HSA becomes a HLSA (Hierarchical *Learning* Stochastic Automaton).

The stochastic approximation reinforcement learning scheme used here at both levels was proposed by Fu[2] for its expediency. First, the reliability of primitive algorithms is estimated by

$$\hat{R}_{ij}(N_{ij}+1) = \hat{R}_{ij}(N_{ij}) + \frac{1}{N_{ij}+1}[z_{ij}(N_{ij}+1) - \hat{R}_{ij}(N_{ij})] \tag{2}$$

which, when $\hat{R}_{ij}(0) = 0$, is no more than the recursive version of the sample mean of the instantaneous performance function $z_{ij} \in \{0, 1\}$, 0 being a penalty (algorithm failed to meet the problem specifications) and 1 a reward. N_{ij} is the number of occurrences of environment state i and algorithm j. It is known that the sample mean estimate converges with probability one (w.p. 1) to the actual value[7]: $\Pr\{\lim_{n\to\infty} \hat{R}_{ij}(n) = R_{ij}\} = 1$. From (2), $\hat{J}_{ij}(N_{ij}) = 1 - \hat{R}_{ij}(N_{ij}) + \rho C_{ij}$. Then, $\hat{J}_{ij}(N_{ij})$ is propagated bottom-up, to be used in the update of task probabilities.

To update the probability density function over the set of tasks or algorithms, Fu's stochastic approximation reinforcement learning algorithm is used:

$$p_{ij}(N_i+1) = p_{ij}(N_i) + \gamma(N_i)(\lambda_{ij}(N_i) - p_{ij}(N_i+1)) \tag{3}$$

where $N_i = \sum_j N_{ij}$, $0 \leq \lambda_{ij}(N_i) \leq 1$, $\sum_j \lambda_{ij}(N_i) = 1$, $i = 1, \ldots, d$ denotes environment states, $j = 1, \ldots, r$ denotes tasks or primitive algorithms. Given the estimates of the performance function at each time instant, $\lambda_{ij}(N_i)$ comes

$$\lambda_{ij}(N_i) = \begin{cases} 1 & \text{if } \hat{J}_{ij}(N_{ij}) = \min_k \hat{J}_{ik}(N_{ik}) \\ 0 & \text{if } \hat{J}_{ij}(N_{ij}) \neq \min_k \hat{J}_{ik}(N_{ik}) \end{cases} \quad (4)$$

We use $\gamma(N_i) = \frac{1}{N_i+1}$, which satisfies the condition of Theorem 2 in [2]. Hence, the probability density function of the action probabilities at state i will converge w.p. 1 to zeros for all the actions except the optimal action u^*, i. e., $\Pr\{\lim_{n\to\infty} \Pr\{u^*|i\} = 1\} = 1$, where u^* for state i is defined as $J_{i*}(N_{i*}) = \min_k\{J_{ik}(N_{ik})\}$, and J_{ij} is the cost function at the decision level under consideration, for state i and algorithm j.

3 Application to an Intelligent Robotic System

A manipulator with 6 DOF has to grasp a strut whose 3D pose (position + orientation) is roughly known. There is a pair of cameras in the ceiling, overviewing the working space of the manipulator and used by a stereo vision system to determine more accurately the 3D pose of the object. The manipulator motion can be position or force controlled. The latter is known as compliant motion. The scene is well illuminated but from time to time lights go off, deteriorating the accuracy of vision algorithms. The environment has 2 states, one corresponding to *lights on* (0) and the other to *lights off* (1). The only command available is $c =$Grab-Strut. The event set E is composed by 5 events:

- e_1 : move-manipulator;
- e_2 : grab-object-compliant;
- e_3 : locate-object;
- e_4 : plan-path;
- e_5 : grab-object-hard.

Event e_1 represents a movement along a pre-planned path. Two algorithms are capable of translating e_1: a Proportional-Integral controller (PI) and a Proportional-Derivative controller (PD). None of them attempts to cancel the non-linear terms in the dynamics of the manipulator, for the sake of simplicity of implementation. Hence, both are prone to errors resulting from incomplete modeling of the manipulator dynamics. Event specifications state that at the end of the movement along the path the end-effector frame of the manipulator must be located within a boundary ϵ_1 of the pose set point - established by the vision system or by the initial rough estimate of the strut pose. Given this specification, the PI controller is more reliable due to its capability of canceling steady state error of the step response of linear systems, and constant disturbances, such as those due to ignoring the nonlinear dynamics when motion is slow.

Event e_2 assumes that the manipulator is close enough to the object to be grasped. The manipulator gripper is opened, and compliantly approaches the object, until the desired 3D force and torque components are obtained. Then, the gripper is closed and the object grasped. Even though the same method is used to accomplish compliant control of the manipulator – Position Accommodation Control (PAC) – different parameterizations of the method result in different algorithms. PAC is a form of position-based force control where the manipulator nominal position can be modified in such a way that it "complies" with external forces. The usual solution to this problem is based on the simulation by the controller of a mechanical impedance of the manipulator to its environment, using mass-spring-damper (MSD) equations. The four algorithms translating e_2 are MSD1, MSD2, MSD3, MSD4, in *increasing* order of stiffness of the damping parameter. The stiffer the PAC the harder is to grasp an object if its location is not accurately known. On the other hand, with a very accurate location estimate, one may desire compliant control, but not too underdamped, so that grasping is more precise. The specification for e_2 requires an absolute steady state force error below some ϵ_2 specification. The reliability increases with stiffness under environment state 0, since a more accurate pose estimation is possible, but decreases with stiffness under state 1.

Event e_3 determines the pose of an object using stereo vision algorithms. The uncertainty on pose determination by stereo vision is mainly due to errors of the matcher when determining image points in the two cameras corresponding to the same world frame point, and pixel resolution which implies that the greater the distance of the object from the cameras, the worst is the depth estimate. Since in this case the camera position is fixed, the first factor prevails. If several frames are taken each time the event is invoked and the matcher works over the resulting average frame, noise will be reduced. The same matcher with an input frame resulting from the average of a different number of frames (N_0 and N_1) corresponds to 2 different algorithms (M_0 and M_1). The specification for e_3 requires that the estimated error statistics when determining the object pose are inside some tolerance interval $[-\epsilon_3, \epsilon_3]$. One such measure could be the estimated variance of the matcher error which increases from state 0 to state 1, because lighting conditions seriously affect the matcher error variance. Hence, if $N_0 < N_1$, reliability will be greater for M_1, but will decrease for both algorithms from state 0 to state 1. Also, if cost is made proportional to the number of averaged samples, M_1 will cost more than M_0 in both states.

Event e_4 plans a path in cartesian space, conveniently samples that path, and using inverse kinematics routines, maps the path into joint space of the manipulator. This path goes to a memory space shared with e_1 - which is part of the World Model, if one exists. e_4 needs an argument stating the destination of the path, given the current departure pose. For a cartesian path planner, statistical fluctuations around the nominal parameters of the manipulator are the sources of uncertainty when the inverse kinematics routines (based on nominal parameters) are called. Only one algorithm is considered here, based on Paul's method for planning a straight line trajectory in cartesian space. Its reliability is considered 100% and the cost is 0.

Event e_5 is a non-compliant version of e_2. All compliance will be passive, i. e., a result of the manipulator mechanical compliance. Specifications are also related to the steady state force error, which is required to be less than $\epsilon_5 = \epsilon_2$. The reliability of the only algorithm used - a PI controller - is necessarily smaller than that of any of the algorithms translating e_2, which provide active compliance, since no force control exists.

Table 1 shows the reliabilities and costs assigned to the different events for simulation purposes. Notice that specifications are made for each event, not for each algorithm. Cost does not change with the environment state. The reliabilities of the algorithms are estimated from the rewards/penalties resulting from successes/failures of their application over the environment, respectively. The algorithm rate of successes/failures is simulated by a Monte Carlo method from the tabulated reliabilities for each pair state/algorithm, and it is unknown to the IM. Alternative tasks differ by the inclusion or not of compliance and by the refining or not of the knowledge of the strut pose. The set of rules R of the stochastic regular grammar $G = (\{e_1, e_2, e_3, e_4, e_5\},\ \{S, A, B, C\}, R, P, S)$ for the command is:

$$
\begin{array}{llll}
0.5 & \mathrm{S} & \rightarrow & e_3\ \mathrm{A} \\
0.5 & \mathrm{S} & \rightarrow & \mathrm{A} \\
1.0 & \mathrm{A} & \rightarrow & e_4\ e_1\ \mathrm{B} \\
0.5 & \mathrm{B} & \rightarrow & e_2\ \mathrm{C} \\
0.5 & \mathrm{B} & \rightarrow & e_5\ \mathrm{C} \\
1.0 & \mathrm{C} & \rightarrow & e_4\ e_1
\end{array}
$$

The numbers to the left of the productions are the initial production probabilities $P(0)$. They were assumed equal for both environment states.

event	e_1		e_2			
algorithm	pd	pi	msd1	msd2	msd3	msd4
specs	$\lvert$ss-error$\rvert < \epsilon_1$		$\lvert$force ss-error$\rvert < \epsilon_2$			
reliab/st0	0.85	0.95	0.80	0.85	0.90	0.95
reliab/st1	0.85	0.95	0.95	0.90	0.85	0.80
cost	0.1	0.1	0.2	0.2	0.2	0.2

event	e_3		e_4	e_5
algorithm	M_0	M_1	Paul's	pi
specs	error var $< \epsilon_3$		-	$\lvert$force ss-error$\rvert < \epsilon_2$
reliab/st0	0.85	0.95	1.00	0.65
reliab/st1	0.70	0.80	1.00	0.65
cost	0.3	0.5	0	0.1

Table 1: Event specifications, translating algorithms and their simulated reliabilities and costs.

All results shown are averages over 50 sample functions of the corresponding stochastic process, for example action probabilities as a function of time. The production probabilities are updated after a task is applied. In the figures, 'iterations' means the number of production probabilities updates. A recursive sample mean with *forgetting factor* was used to improve the adaptiveness of Reliability estimate. Notice however that no proof of convergence exists for the multi-state environment when using the forgetting factor.

Figures 1-2 show an interesting example of adaptation of the IM to an environment change that it does not recognize. At step 400 the environment reliabilities in table 1 are switched from state 0 to state 1 (lights off). Both events e_2 and e_3 suffer dramatic changes, but only the latter influences the production sequence. The change in e_2 is only internal, i. e., when the environment state changes, so does the choice of the best algorithm translating the event. The change in the reliability estimate of e_3 provokes a task change from estimating the pose using stereo to the use of the apriori estimate, as expected, since stereo reliability is poor under bad lighting conditions. Convergence to the optimal actions (algorithms and productions) are observed in state 0.

An usual problem with learning methods is the number of iterations it takes to learn the optimal action. Notice however that since our methodology gives much credit to the designer of the primitive algorithms and tasks, when the actual best action is not the one selected, the incurred error is not greater than the difference in reliability between the most and least reliable tasks. This guarantees a safe convergence to the selection of the optimal task after some runs.

4 Preliminary Conclusions and Future Work

In this paper we formally stated a solution for the problem of Decision Making and Learning in Intel-

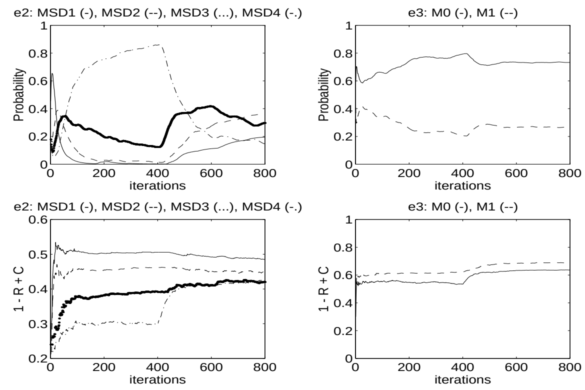

Figure 1: Evolution of probabilities and cost functions for algorithms translating events e_2 and e_3, showing adaptability to a change in the environment at step 400.

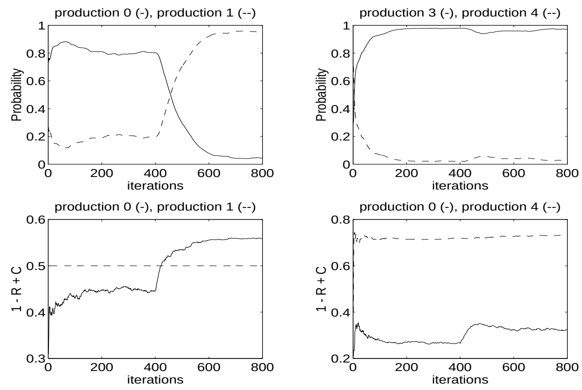

Figure 2: Evolution of probabilities and cost functions for productions 0,1,3 and 4, showing adaptability to a change in the environment at step 400.

ligent Machines, based on a cost function which includes reliability and cost of the involved algorithms at the 3 levels of the hierarchy proposed by Saridis. A case study was presented where the proposed formalism is applied to a robotic system requiring coordination among control, vision and path planning. Preliminary results from a conceptual simulation show the convergence of the reliability estimation and reinforcement scheme algorithms at the different levels of the IM hierarchy, and adaptiveness to changes in state environment.

Future work will consider a more realistic simulation (including manipulator dynamics), increasing the convergence speed of the reinforcement learning algorithms, and an application to process control, to show the flexibility of the method.

Acknowledgments

The first author was supported by the portuguese research institution JNICT, under Grant #BD/1357/91-IA. The second author was supported by the NASA Center for Intelligent Robotic Systems for Space Exploration (CIRSSE) under Grant #NAGW-1333.

References

[1] K. S. Fu and T. L. Booth. Grammatical inference: Introduction and survey – part II. *IEEE Transactions on Systems, Man and Cybernetics*, SMC-5(4), 1975.

[2] K. S. Fu and Z. J. Nikoliç. On some reinforcement techniques and their relation to the stochastic approximation. *IEEE Transactions on Automatic Control*, AC-11(2):756–758, 1966.

[3] P. U. Lima and G. N. Saridis. Measuring Complexity of Intelligent Machines. In *Proceedings of 1993 IEEE Int. Conf. Robotics and Automat.*, may 1993.

[4] P. U. Lima and G. N. Saridis. Performance improvement of Autonomous Underwater Vehicles based on hierachical reinforcement learning. In *Proceedings of 1st ISR Workshop on Autonomous Underwater Vehicles*. Kluwer, 1995.

[5] J. E. McInroy and G. N. Saridis. Entropy searches for robotic reliability assessment. In *Proceedings of 1993 IEEE Int. Conf. Robotic Automat.*, May 1993.

[6] K. S. Narendra and M. A. L. Thathachar. *Learning Automata - an Introduction*. Prentice Hall, 1989.

[7] A. Papoulis. *Probability, Random Variables and Stochastic Processes*. NY:McGraw-Hill, 1965.

[8] Technical Committee on Intelligent Control. Report of task force on Intelligent Control, IEEE Control Systems Society. *IEEE Control Systems Magazine*, 14(3), June 1994. P. Antsaklis, editor.

[9] K. P. Valavanis and G. N. Saridis. *Intelligent Robotic Systems*. Kluwer Publishers, 1992.