LEARNING OF AUTONOMOUS AGENT IN VIRTUAL ENVIRONMENT

Pavel Nahodil, Jaroslav Vítků
CTU in Prague, FEE
Department of Cybernetics
Technicka 2, 16627, Prague 6, Czech Rep.
Email: nahodil@fel.cvut.cz, vitkujar@fel.cvut.cz

KEYWORDS

Agent, Creature, Behavior, Artificial Life, Hybrid Architecture, Hierarchical Approaches, Intentions, Planning

ABSTRACT

Presented topic is from area of development of artificial creatures and proposes new architecture of autonomous agent. The work builds on a research of the latest approaches to Artificial Life, realized by the Department of Cybernetics, CTU in Prague in the last twenty years. This architecture design combines knowledge from *Artificial Intelligence* (AI), Ethology, *Artificial Life* (ALife) and Intelligent Robotics. From the field of classical AI, the fusion of reinforcement learning, planning and artificial neural network into one more complex control system was used here. The main principle of its function is inspired by the field of Ethology, this means that life of given agent tries to be similar to life of an animal in the Nature, where animal learns relatively autonomously from simpler principles towards the more complex ones. The architecture supports on-line learning of all knowledge from the scratch, while the core principle is in hierarchical *Reinforcement Learning* (RL), this action hierarchy is created autonomously based solely on agents interaction with an environment. The main key idea behind this approach is in original implementation of a domain independent hierarchical planner. Our planner is able to operate with behaviors learned by the RL. It means that an autonomously gained hierarchy of actions can be used not only by action selection mechanisms based on the reinforcement learning, but also by a planning system. This gives the agent ability to utilize high-level deliberative problem solving based solely on his experiences. In order to deal with higher-level control rather than a sensory system, the life of agent was simulated in a virtual environment.

INTRODUCTION

One of the oldest dreams of scientist and authors of science-fiction, to create an artificial and intelligent living being, still has not been reached and yet, despite the fact that science makes big steps in this field, after many years of research the mankind is still far from reaching this goal.

At the Department of Cybernetics, we're raising a child machine from nowadays infancy to at least pubescence in a near future - thus bringing Turing's vision to fruition - and creating entirely new approaches to machine learning. In our this research is put effort in a strong behaviorist approach, meaning that we work from principle that language is a skill, not simply the output of brain functions, and, therefore, can be learned. Since long time ago, human have always dreamed to create artificial creature. Nowadays, with the help of technology, this dream nearly came true. Our goal is not simply to build machines that are like humans but to alter our perception of the potential capabilities of robots. Our current attitude toward intelligent robots, we assert, is simply a reflection of our own view of ourselves. We are concerned in a part of Artificial Intelligence where more emphasis is put to the behavior of robots and their interaction with their environment. In this research area, intelligence of robots is evaluated as ability to efficiently exploit the environment in order to meet its goals, this research field is called ALife.

As advancement in the research of robotic system continued, it has been shown that in order to create autonomous robots, too much effort has to be put into low-level functions as is processing data from sensory systems and accurate handling the actuators. So the researches that were interested in more complex types behavior consecutively moved from experiments on real robotic systems towards simulation environments, where the word *robot* was replaced by the word *agent*.

Here will be presented the main today's direction and results of our research in this paper. The recent methods used by us to create intelligent machines will be described here. In later sections will be described how these approaches can be combined together, which results in architectures of autonomous creatures. The main accent is put to ability to learn everything that agent needs from an environment.

USED PRINCIPLES

As it can be seen in the Nature on almost all types of organisms, successful life in our highly complex and dynamic environment requires fusion of more than just one selected approach. This is one of the main reasons,

Proceedings 26th European Conference on Modelling and Simulation ©ECMS Klaus G. Troitzsch, Michael Möhring, Ulf Lotzmann (Editors)
ISBN: 978-0-9564944-4-3 / ISBN: 978-0-9564944-5-0 (CD)

why we have focused on hybrid agent architectures, where a several number of different methods of problem solving and learning are connected together. The most important principles will be described here. These principles are used in order to build our artificial creatures, which exhibit behavior and course of learning similar to real living animals (Nahodil, Kadlecek, D. 2008).

Reinforcement Learning

The key and basic principle is the RL, learning method inspired in behaviorist psychology, where an agent learns, which actions should take in the given state in order to maximize its future reward from his environment. The basic idea is the same with a dynamic programming, it is very general approach and the only main disadvantage is fact that an environment formulated as a *Markov Decision Process* (MDP) is required (Bellman, 1957). Interaction with the MDP environment means that each discrete time step t an agent perceives the finite set of states $|S|$ and is able to execute finite set of actions $|A|$. After executing the selected action u_t in the state x_t, the environment responds with a *reward* or *punishment* $r(x_t, u_t)$ and a *new state* $x_{t+1} = T(x_t, u_t)$ is generated. The next-state function T and the reinforcement r function are not known to the agent, the important property of MDP is that the transition function T is based only on the actual state and executed action.

The goal of RL is to choose actions in response to states so that the reinforcement is maximized, this means that an agent is learning policy: a mapping from states to actions. There are several possible ways to implement a learning process; here was chosen a Q-learning. In this form of RL an agent learns to assign values to state-action pairs a Q-value function, the value of this function is sum of all future events. While immediate rewards are more important, hare is used discounted cumulative reinforcement, where future reinforcements are weighted by value $\gamma \in \langle 0, 1\rangle$. The equation (1) represents the optimal Q-value function.

$$Q^*(x_t, u_t) = r(x_t, u_t) + \gamma \max_{u_{t+1}} Q^*(x_{t+1}, u_{t+1}) \quad (1)$$

At each step, the agent executes one action (selected based on the discounted Q-value function) receives reinforcement and updates Q-value of a given state-action pair in the table according to the off policy *Temporal Difference* (TD) control - equation (2), where $\alpha \in \langle 0, 1\rangle$ is the learning rate.

$$Q(x_t, u_t) \leftarrow Q(x_t, u_t) + \\ + \alpha\big[r(x_t, u_t) + \gamma Q(x_{t+1}, u_{t+1}) - Q(x_t, u_t)\big] \quad (2)$$

In order to get a good trade-off between exploration and exploitation, an action selection mechanism uses some kind of randomization, instead of pure greedy method. A system that implements this entire mechanism will be called *return predictor*.

Hierarchical Reinforcement Learning

The classical RL approach has one disadvantage: size of look-up table (matrix) for storing Q-values grows very fast with an environment complexity. This means that in slightly more complex environment the Q-value matrix can have too many dimensions and the learning convergence can be very slow. In order to beat the course of dimensionality, *Hierarchical RL* (HRL) was introduced. We can define *Decision space* (D) as some defined subset of all possible actions and environment states, over this decision space can operate one return predictor. This decision space can be then seen as an *abstract action*. The main idea of hierarchical RL is very simple: in case of the classical "flat" Q-learning algorithm an agent selects among primitive (one-step) actions. Compared to this, in the hierarchical RL the return predictor can select among primitive and abstract actions (decision spaces). The HRL uses *Semi Markov Decision Process* (SMDP), where a waiting time for the next time step $t + 1$ is random variable.

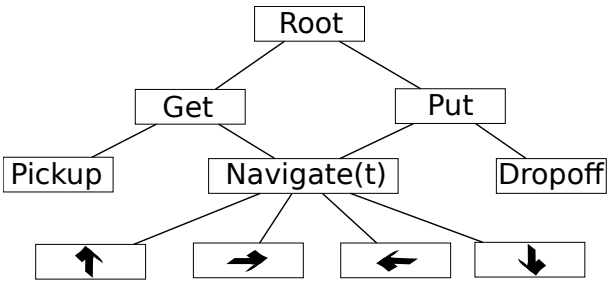

Figure 1: Example of hierarchical task decomposition for well-known taxi problem (Dietterich, 1998)

In our approach is the MAXQ value function decomposition used (Dietterich, 1998), where the received reward can be distributed into a hierarchy of decision spaces D_i using factorization function $\rho(\tilde{r}, D_i)$, where $\tilde{r}$ is the reward generally from the composite behavior. The parameter τ represents positive duration of action, then the Q-learning update formula for decision space D_i is in equation (3).

$$Q_i(x_t, u_t) \leftarrow Q_i(x_t, u_t) + \\ + \alpha\big[\rho(\tilde{r}, D_i) + \gamma^\tau Q_i(x_{t+1}, u_{t+1}) - Q_i(x_t, u_t)\big] \quad (3)$$

Our proposed approach is based on an architecture called *"Hierarchy, Abstraction, Reinforcements, Motivations Agent Architecture"* (HARM) (Kadlecek, 2008). Therefore is used motivation $m(D_i)$ which defines "how much" the agent wants to execute particular action (corresponding to a decision space D_i). The resulting *utility* of action (for a decision space on the top of the hierarchy) is defined in the following equation:

$$\varphi_{D_i}(s,a) = m(D_i)Q_i(s,a). \quad (4)$$

Utilities for the rest of decision spaces in a hierarchy are composed from its own utility and utilities of all parent decision spaces through connection of strength $c_{D_i}^{D_j}(s,a)$ as seen in the equation (5).

$$\varphi_{D_i}(s,a) = m(D_i)Q_i(s,a) \sum_{j \in pars(D_i)} c_{D_i}^{D_j}(s,a)\varphi_{D_j}(s,a) \quad (5)$$

Because of this approach, the motivation to execute a particular behavior (action) can spread through the connection function $c_{D_i}^{D_j}(s,a)$ from the top of a hierarchy towards primitive actions. This means that a selection of concrete primitive action to be executed emerges from various motivations, conditions and dependencies in a whole hierarchy. When a reinforcement/punishment is obtained, this information travels in the opposite direction, from the primitive actions towards the more complex decision spaces on the top of the hierarchy through the factorization function $\rho(\tilde{r}, D_i)$.

Autonomous Creation of RL Hierarchy

Dr. Kadleček, in his Dissertation Thesis, presented HARM system, which is capable of creating such hierarchy of decision spaces autonomously, based on received reinforcements of various types (Kadlecek, 2008). In his architecture, an agent has its own predefined physiology. Agent's physiological state-space, represented by a dynamical system, contains set of agent's internal variables. The physiological state-space contains two important areas: limbo and a purgatory one. Limbo area represents the optimal conditions, if an agent is in this area, no motivation is generated. On the other hand, if an agent is in the purgatory area, an amount of produced motivation increases exponentially. If an agent actively moves some of his physiological variables towards the optimal conditions, a reinforcement is received, if the movement is in another direction, towards the purgatory area, a punishment is received (Kadlecek, Nahodil, P. 2001).

After receiving a reward or a punishment, new decision space D_i in a hierarchy of actions is created and this decision space is connected to a physiological variable through the motivation link $m(D_i)$. Because of this approach, an agent autonomously connects consequences of his behavior with own physiology and learns how to preserve homoeostasis. A set of variables and actions contained in particular decision spaces (and thus also the shape of action hierarchy) is maintained during the agent's life by using four main operations: sub-spacing, behavior associating, variable removing and variable promoting.

PROPOSED NOVEL APPROACH

Later, several improvements in the HARM system were developed, including on-line learning or an intentional state-space. By use of intentions, an agent can autonomously generate its own intentions during his life, this concept will be explained in the following section.

The course of agent's life

At the beginning of the simulation, the agent has no information a priory about world and his regularities and is equipped only with some set of primitive actions and predefined needs, represented as variables in the physiological state-space. These needs can represent for example synonyms for thirst, hunger, or need for recharging a battery.

At first, the agent acts randomly and observes whether something interesting has happened, that is: whether some reward/punishment was received, or whether the agent managed to change some environment variable. In the second case, a new intentional variable (corresponding to the particular ability to influence an environment) is created. In both cases, the new decision space D_i is created and connected to its source of motivation through the motivation link $m(D_i)$.

Later in the simulation, the agent learned some action hierarchy (e.g. he knows how to drink or recharge a battery) and thus exhibits consecutively more and more systematic and stable behavior now.

Compared to agent's predefined physiology, variables in the intentional state-space represent the agent's intention to learn, to "train" some new behavior, autonomously discovered during his interaction with an environment. The intentional state-space has no purgatory area, which causes that the agent learns these behaviors, only in relatively optimal conditions. This concept is similar to learning of young animals by playing with unknown objects.

From Reinforcement Learning towards the Planning

As a latest result of our research in the field of ALife, we have proposed a system that is capable of deliberative "thinking" over this autonomously created hierarchy of abstract actions (decision spaces). This gives the agent whole new dimension of abilities how to use this knowledge.

Compared to the RL, from our point of view, the planning is deliberative approach capable of solving complex tasks, but it requires accurate description of an environment. This requirement can cause problems even in relatively simple environments, where total number of possible states always grows too fast to be handled by a planner. This disadvantage is solved by hierarchical planners, for example *Hierarchical Task Network* (HTN), but these planners are domain dependent, or at least domain configurable. The main advantage of our approach is that our hierarchical planner can beat the course of dimensionality a well as other

hierarchical planners, but moreover maintains its domain independence. In other words our planned is domain self-configurable: by using the autonomous creation of action hierarchy, it can adapt itself to a given domain.

For implementation of planning system was used the world-wide known language, called *Stanford Research Institute Problem Solver* (STRIPS). It is formally represented as a quadruple $\langle P, O, I, G \rangle$. The P is the set of conditions expressed by propositional variables describing the world state, I is the description of initial state and G is description of properties which are fulfilled in a goal state(s). O is the set of operators - actions, each operator consists of the quadruple $\langle \alpha_s, \beta_s, \gamma_s, \delta_s \rangle$. The elements α_s and β_s describe the constraints when the action can be applied, that is: describe which conditions must be true and which false in the given situation. The elements γ_s and δ_s describe action effects after its application, that is: which propositional variables will become true and which false. Roughly speaking, the current state of the world is described by a binary vector, where operators change values of bits on a specified position in a specified manner. The plan is a sequence of applicable operators that consecutively transform the description of initial state towards the state which fulfills the goal conditions.

As a typical planner, STRIPS requires on its input three main things: description of the current state, description of a goal state and a set of possible actions. Our latest architecture, presented in (Vitku, 2011) is able to automatically infers these information from the HARM action hierarchy. This process will be described here in more details.

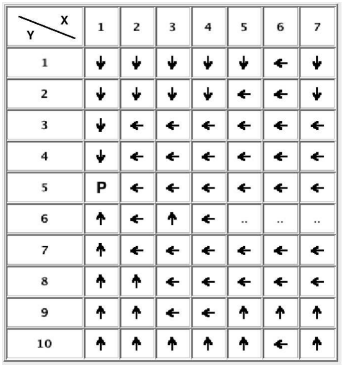

Figure 2: Example of learned behavior which controls the lights. The agent approaches towards the switch and executes action *press* (denoted by P) on the correct position. The successful execution of this behavior switches the value of variable *lights-state* between two possible states: on/off.

In the Fig.2 it can be seen an example of learned decision space represented by a 2D matrix, where each tale corresponds to a position of an agent in the map. Each primitive action (depicted on each tale) represents the learned action, that is the action with the highest Q-value. The agent discovered that by pressing the switch on the left side of the map the light can be switched on/off. It was identified as agent's ability to change some environment property and new intention to learn this behavior was created. The picture represents behavior for turning on/off the lights, which was learned through this motivation. The decision space D_i (matrix of Q-values) contains agent's actual $\langle X, Y \rangle$ position and the variable causing the reinforcement is *lights-state*.

The basic idea is that in order to use this decision space as a primitive action, we need to consider only the "main" variable of a decision space, the variable that **changes during the reinforcement**. In this case, where the decision space consists of three variables: agent's *X* and *Y* position and the *lights-state*, the "main" variable of the decision space is *lights-state*, to the planner will take into account only this variable. Now follows the description of how primitive actions in the STRIPS language are generated: the decision space was created in order to learn the behavior turn on/off the lights. Exactly this does the primitive action in the STRIPS language. In case of a binary variable, this decision space can be represented as two primitive actions in the STRIPS language. The vector describing the problem has one bit **Turn on the lights**, in this simple case only. The action contains precondition: *lights off*, and effect: *lights on*. The action **Turn off the lights** contains precondition: *lights on*, and effect: *lights off*.

The description of entire environment can be automatically generated in form of STRIPS language by use of this principle. The main advantage here (besides the domain independence) is the fact, that only those interesting and potentially important information are passed to the planner. The state description was reduced from 3 variables to one, in the previous example. A hierarchy of RL actions serves here as some kind of filter. This autonomous pre-processor filters information for the deliberative planner, which works over the hierarchy of actions.

SIMULATION RESULTS

For concluding the experiment, the *Massim* simulation environment was used. It was originally created for annual competition called "Multi-agent contest", but for a single-agent case is suitable as well. All information, in both directions agent-environment and environment-agent, is sent through the TCP/IP protocol and coded in the XML format. At each discrete simulation step, the simulator sends all information about an environment to an agent. Agent processes the received data and sends XML with one selected primitive action. Information sent by the environment are in convenient form: for each object in a simulation is send its name and attributes.

In the Fig.3 there is an example of an experiment scenario, the position of lights-switch corresponds to the matrix shown in the Fig.2.

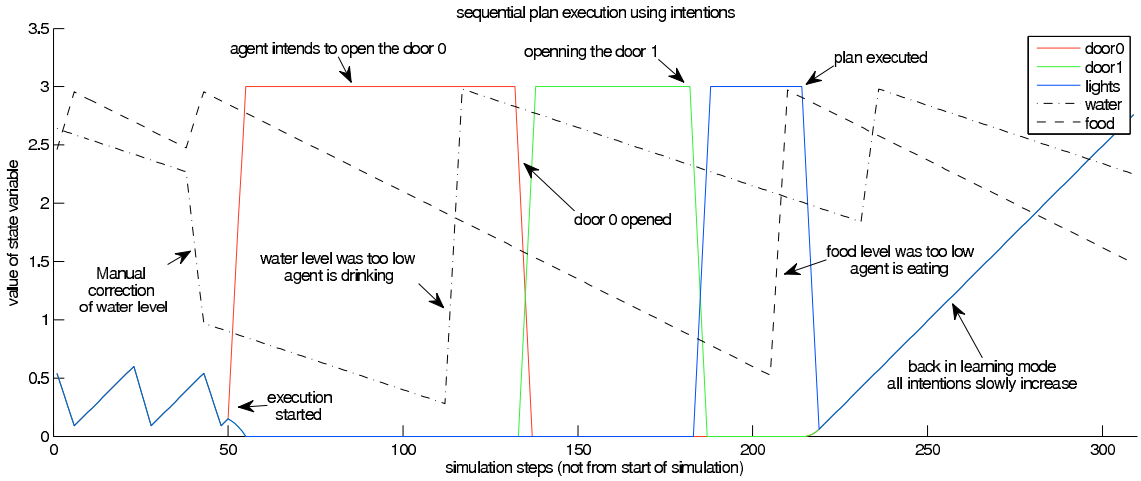

Figure 5: Graph showing the sequential plan execution. The Y axis represents the amount of intention to execute the particular action, the X axis represents time. It can be seen how the planner consequently set the intention to *open the door0*, *open the door1* and *turn on the lights*. The dashed lines represent amounts of *water* and *food* in the agents body. The *water* and *food* levels in the agents body fall towards zero with time, in the picture there are results of *drinking* and *eating* as refilling these amounts towards the maximum.

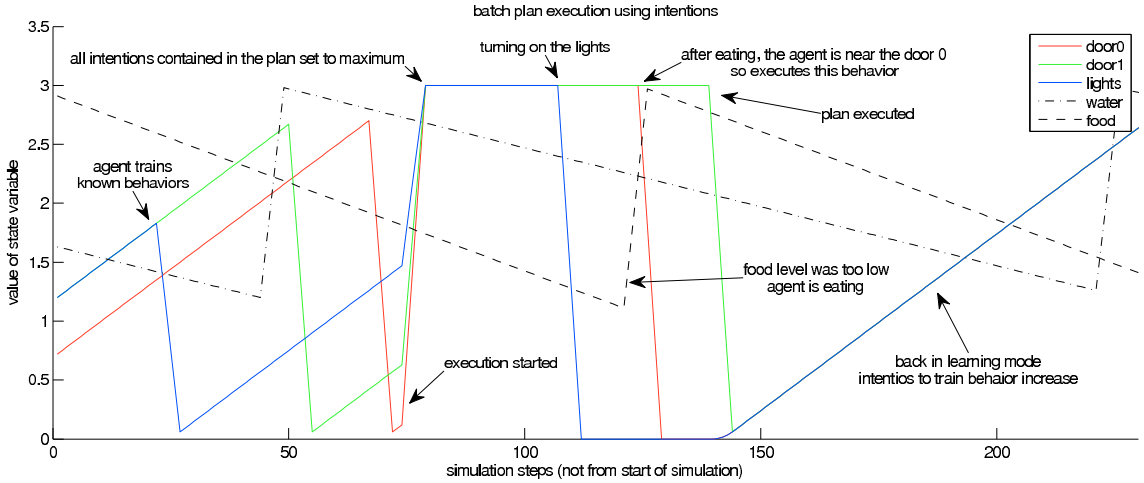

Figure 6: Graph showing the batch plan execution. It can be seen how the planner set intentions corresponding to all actions in the plan to the maximum and waited until all intentions falls towards the zero, which means that the plan was executed. The dashed lines represent amounts of *water* and *food* in the agents body.

Selected Experiment

The agents ability to autonomously identify potentially useful behaviors was tested in our selected experiment. It means, to learn how to open the door and switch the lights. Tested ability to represent these behaviors as a set of primitive actions in the STRIPS language and use them for planning and plan execution was also verified here.

The picture Fig.4 shows three autonomously learned behaviors as decision spaces in form of RL. The meaning of the matrixes is described under this figure. These 3 actions were autonomously represented as 6 primitive actions in the STRIPS language.

Reduction of Decision State Space Size

The original description of the world that contained $10 \times 10 \times 2 \times 2 \times 2 = 800$ states was reduced to the world description for the planning engine to only $2 \times 2 \times 2 = 8$ states, represented as a binary vector with 3 variables. Here can be seen autonomous and effective preprocessing information about the environment to the higher-level decision making system, where the planner does not have to care about the agent's actual position.

The benefit provided by this presented method of state space size reduction dramatically increases with a complexity of given domain. This was tested in other examples where the resulting reduction of decision space was from 64×10^{12} to only 100 states (Vitku, 2011).

For testing agent's ability to create a plan and execute it, the user wants to enable passing through the hallway

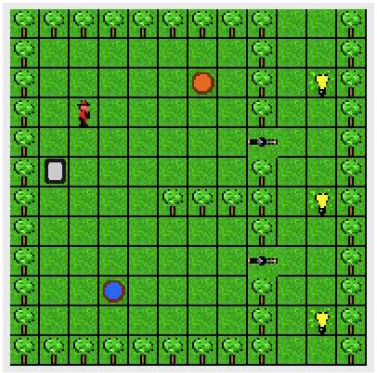

Figure 3: Map of the environment containing one red agent, silver *lights-switch*, blue source of water, orange source of food and two buttons controlling doors. The hallway on the right contains yellow lights and two doors that can be opened/closed. The map corresponds to the learned strategy depicted in the Fig.2

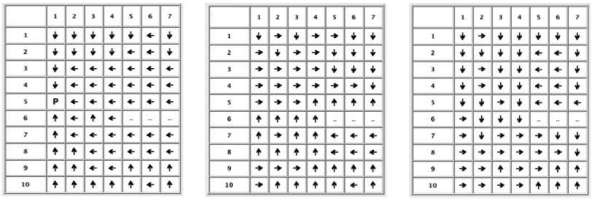

Figure 4: The result of agent's autonomous learning based on the autonomously generated intentions. After discovering that the switch controls lights and the two other switches control the door, these three new decision spaces were created. On the left there is behavior corresponding to lights-control, the center and right matrix represents behaviors corresponding to door control. Note: the door are controlled by approaching to the switch.

on the right side of the map. This requires that the lights are on and both doors are opened (see Fig.3). Actual state is that lights are off and doors are closed. After specifying the goal, the agent was able to create plan containing the sequence of these actions: *open the door0*, *open the door1* and *turn on the lights*. In the following section shows two main ways how a plan can be executed by the planning engine.

Sequential Plan Execution

The first possibility is to execute a given plan sequentially. The Fig.5 depicts this type of plan execution. The planner can influence agent's behavior by manual setting the values of intentions for execution of particular actions. After successful execution of an action, the intention to execute an action falls towards the zero. The planner in this sequential mode sets the intention corresponding to the first action in the plan and waits until it falls towards the zero, after that, the intention to execute next action is set to maximum.

Batch Plan Execution

Now, let's consider simpler problems, where a depth of action hierarchy is not too big and it can be shown that autonomous creation of primitive STRIPS actions can be accurate. Also another type of plan execution can be used here: so called batch mode, after this assumption. This type of plan execution exploits much better the benefits of hierarchical RL. The agent selects an order of actions according to the most efficient strategy, in the current situation. The obvious disadvantage of this type of plan execution requires that the order of actions can be arbitrary.

The Fig.6 depicts how the planner set all intentions contained in the plan to the maximum and waited until all fall towards zero. The main benefit of this approach is in the fact that order of actions to be executed is based on the actual conditions. From the graph it can be seen that the agent was obviously near the *lights-switch* at the beginning, so the maximum utility was produced by the *lights-switch* behavior. After turning on the lights, corresponding utility fell to zero and the next best action was selected.

A good example of exploiting the main benefit of this approach is shown here. After execution of the first action (*turn on the lights*) the *food level* in the agents body was too low and thus the agent had to eat. After eating, the agent was near the water source, which is near the *door0 control*. Obviously the best strategy is to open the *door0*, in this situation.

The Fig.6 also shows that this execution of the same plan was much faster than in the previous-sequential case.

CONCLUSION

We are going to create more sophisticated algorithm with ability to infer preconditions and effects for general problems, in the near future. This includes the ability to infer high-level variables with much more than two states. Also, the algorithm should be used also in domains where effects of particular actions in a hierarchy could interfere. Also, we would like to conclude some experiments in a real environment with some more complicated sensory data. Currently we work on finding of possible real applications of our novel approach in daily praxis.

The most of nowadays similar architectures (which incorporate some hierarchical structures) are domain dependent. This means that some form of domain description has to be specified before the start of any simulation. In case of the commonly used *Belief-Desire Intention Architecture* (BDI) (Sardina, et. al. 2006), the main disadvantage is the plan library. This library describes a causalities in an environment and has to be predefined by user. The another similar idea: an agent architecture which use combination of RL and planning, can be seen in the work called *Reinforcement-Learning Teleo-Operators* (RL-TOPs) (Ryan, Pendrith 1998). In this approach each high-level action (represented by a predicate: e.g. *move(a,b)*) is implemented by some primitive behavior in form of RL. While primitive

behaviors are learned autonomously by the RL engines, the high-level preconditions and effects of actions have to be predefined by the designer.

We believe that, compared to similar ideas found by us, the main advantages of our architecture are:

- The fact that an agent is able of *completely unsupervised adaptation* to a given domain.
- Here, an user does not have to predefine (and therefore he *does not have to know) any part of problem structure*, because our system is able to determine the important knowledge itself, store it and provide it for later reuse.
- The autonomously created hierarchy of actions provides *huge reduction of searched decision space*. In situations where the classical planners fail and hierarchical planners need help of designer, our agent is still able to operate without any problems.

ACKNOWLEDGEMENT

This research has been funded by the Dept. of Cybernetics, Faculty of Electrical Engineering, Czech Technical University in Prague and Centre for Applied Cybernetics under Project 1M0567.

REFERENCES

Bellman, R. (1957), *A Markovian Decision Process*. Indiana Univ. Math. J. 6: pages 679–684.

Dietterich, T.G. (1998), Hierarchical reinforcement learning with the maxq value function decomposition. *Journal of Artificial Intelligence Research 13:* pages 227–303.

Kadleček, D. (2008), *Motivation Driven Reinforcement Learning and Automatic Creation of Behavior Hierarchies*. PhD thesis supervised by Nahodil, P., CTU in Prague, FEE, dept. of Cybernetics, pp. 134, Prague

Kadleček, D., Nahodil, P. (2001), New Hybrid Architecture in Artificial Life Simulation. *In: Proc. of 6th European Conf. of Artificial Life: Advances in Artificial Life*. ECAL 2001, Prague, LNAI Nr. 2159, vol. 1, Printed by Springer, Berlin, vol.1, pages 143-146

Nahodil, P., Kadlecek, D. (2008), Adopting Animal Concepts in Hierarchical Reinforcement Learning and Control of Intelligent Agents. *In Proc. of 2nd IEEE/RAS-EMBS International Conf. on Biomedical Robotics and Biomechatronics,* BioRob 2008, Scottsdale, U.S.A, 2008, pages 122-131

Vítků, J. (2011). *An Artificial Creature Capable of Learning from Experience in Order to Fulfill More Complex Tasks*. Diploma thesis supervised by Nahodil, P., CTU in Prague, FEE, Dept. of Cybernetics, Prague, pp. 123

Ryan M. and Pendrith, M. (1998), RL-TOPs: An Architecture for Modularity and Re-Use in Reinforcement Learning *In Proceedings of the Fifteenth International Conference on Machine Learning,* San Francisco, CA, USA, pages 481-487

Sardina, S., de Silva, L., Padgham, L. (2006), Hierarchical Planning in BDI Agent Programming Language: a Formal Approach. *AAMAS 06 Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems,* ACM New York, pages 1001-1008

AUTHOR BIOGRAPHIES

PAVEL NAHODIL was born in Prague, Czech Republic. Since 1986 he has been a Professor of Technical Cybernetics at the Department of Cybernetics at the Faculty of Electrical Engineering, CTU in Prague. His present professional interest includes artificial intelligence, multi-agent systems, intelligent robotics (control systems of humanoids) and artificial life approaches in general. He is (co-)author of several books, university lecture notes, hundreds of scientific papers and some collection of scientific studies. He is also the international conferences organizer + reviewer (IPC Member) and a member of many Editorial Boards. His e-mail address is: nahodil@fel.cvut.cz

JAROSLAV VÍTKŮ was born in Prague, Czech Republic, graduated in 2011 in Czech Technical University in Prague, Faculty of Electrical Engineering in Artificial Intelligence. His diploma thesis was awarded by Price of Dean. Currently is a PhD student in the CTU, FEE, Department of Cybernetics. His research interest includes behavioral robotics, cognitive science, biologically inspired algorithms and Artificial Life in common. His e-mail address is: vitkujar@fel.cvut.cz